



iSeries

ILE C/C++ for iSeries
Run-Time Library Functions



IBM

@server

iSeries

ILE C/C++ for iSeries
Run-Time Library Functions

Version 5

SC41-5607-01

Note

Before using this information and the product it supports, be sure to read the information in "Appendix B. Notices" on page 495. Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change or addition.

Second Edition (September 2002)

This edition applies to version 5, release 2, modification 0 of ILE C/C++ for iSeries Run-Time Library Functions (product number 5769-CX2), and to all subsequent releases and modifications until otherwise indicated in new editions. This edition applies only to reduced instruction set computer (RISC) systems.

This edition replaces SC09-5607-00.

© Copyright International Business Machines Corporation 1999, 2002. All rights reserved. US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Tables vii	Type Conversion
	Conversion
About ILE C/C++ for iSeries Run-time	Record Input/Output
Library Functions (SC41-5607) ix	Stream Input/Output
	Handling Argument Lists
Who should read this book ix	Pseudorandom Numbers
A note about examples ix	Dynamic Memory Management 31
iSeries Navigator ix	Memory Objects
Installing iSeries Navigator x	Environment Interaction
Prerequisite and related information xi	String Operations
How to send your comments xi	Character Testing
	Multibyte Character Testing
Part 1. Run-Time Library Functions 1	Character Case Mapping
•	Multibyte Character Manipulation
Chapter 1. Include Files 3	Data Areas
<assert.h></assert.h>	Message Catalogs
<pre><ctype.h></ctype.h></pre>	Regular Expression
<pre><decimal.h></decimal.h></pre>	abort() — Stop a Program
<pre><errno.h></errno.h></pre>	abs() — Calculate Integer Absolute Value 38
<pre><except.h></except.h></pre>	acos() — Calculate Arccosine
<pre><float.h></float.h></pre>	asctime() — Convert Time to Character String 40
<pre><langinfo.h></langinfo.h></pre>	asctime_r() — Convert Time to Character String
<pre></pre>	(Restartable)
<pre><locale.h></locale.h></pre>	asin() — Calculate Arcsine
<math.h></math.h>	assert() — Verify Condition 44
<pre><mailocinfo.h></mailocinfo.h></pre>	atan() – atan2() — Calculate Arctangent 45
<pre><monetary.h></monetary.h></pre>	atexit() — Record Program Ending Function 46
<nl_types.h></nl_types.h>	atof() — Convert Character String to Float 47
<pre><pre><pre><pre><pre><pre><pre><pre></pre></pre></pre></pre></pre></pre></pre></pre>	atoi() — Convert Character String to Integer 49
<pre><pointer.h></pointer.h></pre>	atol() — atoll() — Convert Character String to Long
<regex.h></regex.h>	or Long Long Integer 50
<pre><setimp.h></setimp.h></pre>	Bessel Functions
<signal.h></signal.h>	bsearch() — Search Arrays
<stdarg.h></stdarg.h>	btowc() — Convert Single Byte to Wide Character 54
<pre><stddef.h></stddef.h></pre>	_C_Get_Ssn_Handle() — Handle to C Session 55
<stdio.h></stdio.h>	calloc() — Reserve and Initialize Storage 56
<stdlib.h></stdlib.h>	catclose() — Close Message Catalog 57
<string.h></string.h>	catgets() — Retrieve a Message from a Message
<strings.h></strings.h>	Catalog
<time.h></time.h>	catopen() — Open Message Catalog 59
<wchar.h></wchar.h>	ceil() — Find Integer >= Argument 61
<wcstr.h></wcstr.h>	clearerr() — Reset Error Indicators
<wctype.h></wctype.h>	clock() — Determine Processor Time
<xxcvt.h></xxcvt.h>	cos() — Calculate Cosine
<xxdtaa.h></xxdtaa.h>	cosh() — Calculate Hyperbolic Cosine
<xxenv.h></xxenv.h>	ctime() — Convert Time to Character String 66
<xxfdbk.h></xxfdbk.h>	ctime_r() — Convert Time to Character String
Machine Interface (MI) Include Files	(Restartable)
	difftime() — Compute Time Difference
Chapter 2. Library Functions 21	div() — Calculate Quotient and Remainder 70
The C Library	erf() – erfc() — Calculate Error Functions
Error Handling	exit() — End Program
Searching and Sorting	exp() — Calculate Exponential Function
Mathematical	fabs() — Calculate Floating-Point Absolute Value 74
Time Manipulation	fclose() — Close Stream
11110 1111111 pallatori	fdopen() — Associates Stream With File Descriptor 76

|

	feof() — Test End-of-File Indicator 79	_ltoa - Convert Long Integer to String	167
	ferror() — Test for Read/Write Errors	longjmp() — Restore Stack Environment	168
	fflush() — Write Buffer to File 80	malloc() — Reserve Storage Block	170
	fgetc() — Read a Character	mblen() — Determine Length of a Multibyte	
	fgetpos() — Get File Position	Character	172
	fgets() — Read a String	mbrlen() — Determine Length of a Multibyte	
	fgetwc() — Read Wide Character from Stream 86	Character (Restartable)	172
			173
	fgetws() — Read Wide-Character String from Stream 88	mbrtowc() — Convert a Multibyte Character to a	177
	fileno() — Determine File Handle 90	Wide Character (Restartable)	
	floor() —Find Integer <=Argument 91	mbsinit() — Test State Object for Initial State	179
	fmod() — Calculate Floating-Point Remainder 91	mbsrtowcs() — Convert a Multibyte String to a	
	fopen() — Open Files	Wide Character String (Restartable)	180
	fprintf() — Write Formatted Data to a Stream 99	mbstowcs() — Convert a Multibyte String to a	
	fputc() — Write Character	Wide Character String	182
	_fputchar - Write Character	mbtowc() — Convert Multibyte Character to a	
	fputs() — Write String	Wide Character	186
	fputwc() — Write Wide Character	memchr() — Search Buffer	
	fputws() — Write Wide-Character String 106	memcmp() — Compare Buffers	
	fread() — Read Items	memcpy() — Copy Bytes	
	free() — Release Storage Blocks	memicmp - Compare Bytes	190
	freopen() — Redirect Open Files	memmove() — Copy Bytes	191
	frexp() — Separate Floating-Point Value 112	memset() — Set Bytes to Value	192
	fscanf() — Read Formatted Data	mktime() — Convert Local Time	
	fseek() — fseeko() — Reposition File Position 114	modf() — Separate Floating-Point Value	
	fsetpos() — Set File Position	nl_langinfo() —Retrieve Locale Information	
ı			
	ftell() — ftello() — Get Current Position	perror() — Print Error Message	100
	fwide() — Determine Stream Orientation 120	pow() — Compute Power	
	fwprintf() — Format Data as Wide Characters and	printf() — Print Formatted Characters	
	Write to a Stream	putc() – putchar() — Write a Character	
	fwrite() — Write Items	putenv() — Change/Add Environment Variables	
	fwscanf() — Read Data from Stream Using Wide	puts() — Write a String	212
	Character	putwc() — Write Wide Character	
	gamma() — Gamma Function	putwchar() — Write Wide Character to stdout	
	_gcvt - Convert Floating-Point to String 132	qsort() — Sort Array	
	getc() – getchar() — Read a Character	QXXCHGDA() —Change Data Area	
	getenv() — Search for Environment Variables 135	QXXDTOP() — Convert Double to Packed Decimal	
		- 0	
	_GetExcData() — Get Exception Data	QXXDTOZ() —Convert Double to Zoned Decimal	220
	gets() — Read a Line	QXXITOP() —Convert Integer to Packed Decimal	221
	getwc() — Read Wide Character from Stream 138	QXXITOZ() —Convert Integer to Zoned Decimal	221
	getwchar() — Get Wide Character from stdin 140	QXXPTOD() —Convert Packed Decimal to Double	222
	gmtime() — Convert Time	QXXPTOI() —Convert Packed Decimal to Integer	223
	gmtime_r() — Convert Time (Restartable) 143	QXXRTVDA() —Retrieve Data Area	223
	hypot() — Calculate Hypotenuse	QXXZTOD() —Convert Zoned Decimal to Double	224
	isascii() — Test for ASCII Value	QXXZTOI() —Convert Zoned Decimal to Integer	225
	isalnum() - isxdigit() — Test Integer Value	raise() — Send Signal	
ı			
	isblank() — Test for Blank or Tab Character 149	V' = V	227
	iswalnum() to iswxdigit() — Test Wide Integer	_Racquire() —Acquire a Program Device	
	Value		229
	iswctype() — Test for Character Property 152	_Rcommit() —Commit Current Record	230
	_itoa - Convert Integer to String	_Rdelete() —Delete a Record	232
	labs() — llabs() — Calculate Absolute Value of	_Rdevatr() —Get Device Attributes	
	Long and Long Long Integer	realloc() — Change Reserved Storage Block Size	234
	ldexp() — Multiply by a Power of Two	9	237
		regerror() — Return Error Message for Regular	201
	ldiv() — lldiv() — Perform Long and Long Long Division 156		220
	Division	Expression	238
	localeconv() — Retrieve Information from the	regexec() — Execute Compiled Regular Expression	240
	Environment	regfree() — Free Memory for Regular Expression	242
	localtime() — Convert Time		243
	localtime_r() — Convert Time (Restartable) 164	rename() — Rename File	244
	log() — Calculate Natural Logarithm 165	rewind() — Adjust Current File Position	245
	log100 — Calculate Base 10 Logarithm	Rfeod() —Force the End-of-Data.	246

_Rfeov() —Force the End-of-File	247	strncat() — Concatenate Strings	343
_Rformat() —Set the Record Format Name		strncmp() — Compare Strings	
_Rindara() —Set Separate Indicator Area		strncpy() — Copy Strings	
_Riofbk() —Obtain I/O Feedback Information		strnicmp - Compare Substrings Without Case	
_Rlocate() —Position a Record		Sensitivity	347
_Ropen() — Open a Record File for I/O Operations		strpbrk() — Find Characters in String	348
_Ropnfbk() — Obtain Open Feedback Information		strnset - strset - Set Characters in String	
_Rpgmdev() — Set Default Program Device		strptime()— Convert String to Date/Time	
_Rreadd() — Read a Record by Relative Record		strrchr() — Locate Last Occurrence of Character in	
Number	263	String	354
_Rreadf() — Read the First Record		strspn() —Find Offset of First Non-matching	001
_Rreadindv() — Read from an Invited Device		Character	355
_Rreadk() — Read a Record by Key		strstr() — Locate Substring	
_Rreadl() — Read the Last Record		strtod() — Convert Character String to Double	
_Rreadn() — Read the Next Record		strtok() — Tokenize String	
_Rreadnc() — Read the Next Record :	213	strtok() — Tokenize String (Restartable)	
	279	strtol() — strtoll() — Convert Character String to	301
Subfile			262
		Long and Long Long Integer strtoul() — strtoull() — Convert Character String to	302
_Rreads() — Read the Same Record			264
_Rrelease() — Release a Program Device		Unsigned Long and Unsigned Long Long Integer .	
_Rrlslck() — Release a Record Lock	283	strxfrm() — Transform String	300
_Rrollbck() — Roll Back Commitment Control	206	swprintf() — Format and Write Wide Characters to	267
Changes	286	Buffer	
_Rupdate() — Update a Record	288	swscanf() — Read Wide Character Data	
_Rupfb() — Provide Information on Last I/O	200	system() — Execute a Command	
Operation		tan() — Calculate Tangent	
_Rwrite() — Write the Next Record		tanh() — Calculate Hyperbolic Tangent	
_Rwrited() — Write a Record Directly		time() — Determine Current Time	
_Rwriterd() — Write and Read a Record	296	tmpfile() — Create Temporary File	
_Rwrread() — Write and Read a Record (separate		tmpnam() — Produce Temporary File Name	
buffers)		toascii() — Convert Character	
scanf() — Read Data		V 11 V	377
setbuf() — Control Buffering		towctrans() — Translate Wide Character	378
setjmp() — Preserve Environment		towlower() -towupper() — Convert Wide	
setlocale() — Set Locale		Character Case	
setvbuf() — Control Buffering		_ 0 0 0	380
signal() — Handle Interrupt Signals		ungetc() — Push Character onto Input Stream	381
sin() — Calculate Sine	315	ungetwc() — Push Wide Character onto Input	
sinh() — Calculate Hyperbolic Sine		Stream	382
sprintf() — Print Formatted Data to Buffer		va_arg() - va_end() - va_start() — Access Function	
sqrt() — Calculate Square Root		Arguments	
srand() — Set Seed for rand() Function		vfprintf() — Print Argument Data to Stream	386
snprintf() — Print Formatted Data to Buffer		vfwprintf() — Format Argument Data as Wide	
sscanf() — Read Data	322	Characters and Write to a Stream	
strcasecmp() — Compare Strings without Case		vprintf() — Print Argument Data	389
Sensitivity		vsnprintf() — Print Argument Data to Buffer	390
strcat() — Concatenate Strings		vsprintf() — Print Argument Data to Buffer	391
strchr() — Search for Character	325	vswprintf() — Format and Write Wide Characters	
strcmp() — Compare Strings		to Buffer	393
strcmpi - Compare Strings Without Case Sensitivity	328	vwprintf() — Format Argument Data as Wide	
strcoll() — Compare Strings	329	Characters and Print	394
strcpy() — Copy Strings	330	wcrtomb() — Convert a Wide Character to a	
strcspn() — Find Offset of First Character Match	331	Multibyte Character (Restartable)	396
strdup - Duplicate String	332	wcscat() — Concatenate Wide-Character Strings	400
strerror() — Set Pointer to Run-Time Error Message	333	wcschr() — Search for Wide Character	402
strfmon() — Convert Monetary Value to String		wcscmp() — Compare Wide-Character Strings	
strftime() — Convert Date/Time to String			404
stricmp - Compare Strings without Case Sensitivity		wcscpy() — Copy Wide-Character Strings	
strlen() — Determine String Length		wcscspn() — Find Offset of First Wide-Character	
strncasecmp() — Compare Strings without Case		Match	406
Sensitivity	341	wcsftime() — Convert to Formatted Date and Time	

I

| |

wcslen() — Calculate Length of Wide-Character	wmemcmp() —Compare Wide-Character Buffers 443
String	wmemcpy() —Copy Wide-Character Buffer 444
wcsncat() — Concatenate Wide-Character Strings 410	wmemmove() — Copy Wide-Character Buffer 445
wcsncmp() — Compare Wide-Character Strings 411	wmemset() — Set Wide Character Buffer to a Value 446
wcsncpy() — Copy Wide-Character Strings 413	wprintf() — Format Data as Wide Characters and
wcsicmp — Compare Wide Character Strings	Print
without Case Sensitivity	wscanf() — Read Data Using Wide-Character
_wcsnicmp — Compare Wide Character Strings	Format String
without Case Sensitivity	Ü
wcspbrk() — Locate Wide Characters in String 416	Chapter 3. Run-Time Considerations 451
wcsrchr() — Locate Last Occurrence of Wide	errno Macros
Character in String 417	errno Values for Integrated File System Enabled C
wcsrtombs() — Convert Wide Character String to	Stream I/O
Multibyte String (Restartable)	Record Input and Output Error Macro to Exception
wcsspn() — Find Offset of First Non-matching	
Wide Character	Mapping
wcsstr() — Locate Wide-Character Substring 421	Signal Handling Action Definitions
wcstod() — Convert Wide-Character String to	Signal to iSeries Exception Mapping 458
Double	Cancel Handler Reason Codes
wcstol() — wcstoll() — Convert Wide Character	Exception Classes
String to Long and Long Long Integer 424	Data Type Compatibility
wcstombs() — Convert Wide-Character String to	Run-time Character Set
Multibyte String	Asynchronous Signal Model 470
wcstok() — Tokenize Wide-Character String 429	
wcstoul() — wcstoull() — Convert WideCharacter	Part 2. Appendixes 473
String to Unsigned Long and Unsigned Long Long	
Integer	Appendix A. Library Functions and
wcswcs() — Locate Wide-Character Substring 432	Extensions 475
wcswidth() — Determine the Display Width of a	
Wide Character String	Standard C Library Functions Table, By Name 475
wcsxfrm() — Transform a Wide-Character String 434	ILE C Library Extensions to C Library Functions
wctob() — Convert Wide Character to Byte 435	Table
wctomb() — Convert Wide Character to Multibyte	
Character	Appendix B. Notices 495
wctrans() —Get Handle for Character Mapping 437	Programming Interface Information 497
wctype() — Get Handle for Character Property	Trademarks
Classification	
wcwidth() — Determine the Display Width of a	Bibliography 499
	9,
Wide Character	Index 503
wfopen() —Open Files	IIIUGA
Wide-Character Buffer 442	

Tables

	1.	Grouping Example	20.	Exception Classes	. 461
I	2.	Monetary Formatting Example 161	21.	ILE C Data Type Compatibility with ILE RPG	
I	3.	Monetary Fields	22.	ILE C Data Type Compatibility with ILE	
	4.	Values of Precision 206		COBOL	. 463
	5.		23.	ILE C Data Type Compatibility with ILE CL	465
	6.	Return values of strcasecmp() 323	24.	ILE C Data Type Compatibility with OPM	
	7.	Flags		71 1 7	. 465
	8.	Conversion Characters	25	ILE C Data Type Compatibility with OPM	. 100
	9.	Return values of strncasecmp()	20.	COBOL/400	466
ı	10.	Return values of _wcsicmp()	26.	ILE C Data Type Compatibility with CL	467
i	11.	Return values of wcsicmp()	27.		107
'	12.	errno Macros	27.	CL Call to an ILE C Program	160
			20		. 400
	13.	errno Values for IFS Enabled C Stream I/O 453	28.	CL Constants Passed from a Compiled CL	1.00
	14.	Record Input and Output Error Macro to		Program to an ILE C Program	. 468
		Exception Mapping 455	29.	CL Variables Passed from a Compiled CL	
	15.	Handling Action Definitions for Signal Values 456		Program to an ILE C Program	. 468
ı	16.	Default Actions for Signal Values 457	30.	Invariant Characters	. 469
	17.	Signal to iSeries Exception Mapping 458	31.	Variant Characters in Different CCSIDs	469
	18.	Determining Canceled Invocation Reason	32.	Standard C Library Functions	. 475
		Codes	33.	ILE C Library Extensions	
	19.	Common Reason Code for Cancelling		,	
		Invocations 460			

About ILE C/C++ for iSeries Run-time Library Functions (SC41-5607)

This book provides reference information about:

- Include files
- · Run-time functions
- Run-time considerations

Use this book as a reference when you write Integrated Language Environment (ILE) C applications.

This book does not describe how to program in the C programming language or explain the concepts of ILE. Companion publications for this reference are:

- ILE C for AS/400 Language Reference
- WebSphere Development Studio: ILE C/C++ Programmer's Guide
- ILE Concepts
- ILE C/C++ for AS/400 MI Library Reference

For information about other ILE C publications, see the iSeries Information Center at http://www.ibm.com/eserver/iseries/infocenter.

For a list of related publications, see "Bibliography" on page 499.

Who should read this book

This book is intended for programmers who are familiar with the C programming language and who want to write or maintain ILE C applications. You must have experience in using applicable iSeries menus, and displays or control language (CL) commands. You also need knowledge of Integrated Language Environment as explained in the *ILE Concepts* manual.

A note about examples

The examples in this book that illustrate the use of library functions are written in a simple style. The examples do not demonstrate all possible uses of C language constructs. Some examples are only code fragments and do not compile without additional code.

All complete runnable examples for library functions and machine interface instructions are in library QCPPLE, in source file QACSRC. Each example name is the same as the function name or instruction name. For example, the source code for the example illustrating the use of the <code>Rcommit()</code> function in this book is in library QCPPLE, file QACSRC, member RCOMMIT. The QSYSINC library must be installed.

iSeries Navigator

iSeries Navigator is a powerful graphical interface for Windows clients. With iSeries Navigator, you can manage and administer your iSeries systems from your Windows desktop.

You can use iSeries Navigator to manage communications, printing, database, security, and other system operations. iSeries Navigator includes Management Central for managing multiple iSeries systems centrally.

Figure 1 shows an example of the iSeries Navigator display:

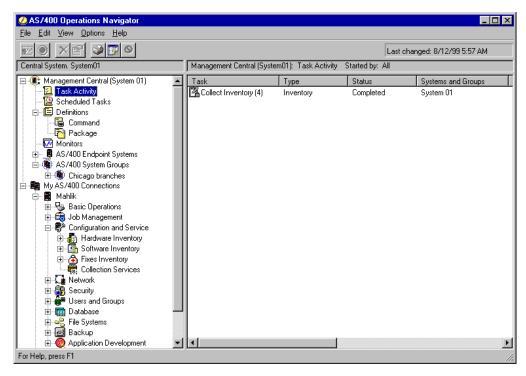


Figure 1. iSeries Navigator Display

This new interface has been designed to make you more productive and is the only user interface to new, advanced features of OS/400. Therefore, IBM recommends that you use iSeries Navigator, which has online help to guide you. While this interface is being developed, you may still need to use a traditional emulator such as PC5250 to do some of your tasks.

Installing iSeries Navigator

To use iSeries Navigator, you must have iSeries Access installed on your Windows PC. For help in connecting your Windows PC to your iSeries system, consult *iSeries Access for Windows--Setup*, SC41-5507-03.

iSeries Navigator is a separately installable component of iSeries Access that contains many subcomponents. If you are installing for the first time and you use the **Typical** installation option, the following options are installed by default:

- iSeries Navigator base support
- Basic operations (messages, printer output, and printers)

To select the subcomponents that you want to install, select the **Custom** installation option. (After iSeries Navigator has been installed, you can add subcomponents by using iSeries Access Selective Setup.)

- 1. Display the list of currently installed subcomponents in the **Component Selection** window of **Custom** installation or Selective Setup.
- 2. Select iSeries Navigator.

3. Select any additional subcomponents that you want to install and continue with **Custom** installation or Selective Setup.

After you install iSeries Access, double-click the iSeries Navigator icon on your desktop to access iSeries Navigator and create an iSeries connection.

Prerequisite and related information

Use the iSeries Information Center as your starting point for looking up iSeries technical information.

You can access the Information Center two ways:

- From the following Web site: http://www.ibm.com/eserver/iseries/infocenter
- From CD-ROMs that ship with your Operating System/400 order: iSeries Information Center, SK3T-4091-02. This package also includes the PDF versions of iSeries manuals, iSeries Information Center: Supplemental Manuals, SK3T-4092-01, which replaces the Softcopy Library CD-ROM.

The iSeries Information Center contains advisors and important topics such as Java, TCP/IP, Web serving, secured networks, logical partitions, clustering, CL commands, and system application programming interfaces (APIs). It also includes links to related IBM Redbooks and Internet links to other IBM Web sites such as the Technical Studio and the IBM home page.

With every new hardware order, you receive the iSeries Setup and Operations CD-ROM, SK3T-4098-01. This CD-ROM contains IBM @server iSeries Access for Windows and the EZ-Setup wizard. iSeries Access offers a powerful set of client and server capabilities for connecting PCs to iSeries servers. The EZ-Setup wizard automates many of the iSeries setup tasks.

For other related information, see the "Bibliography" on page 499.

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or any other iSeries documentation, fill out the readers' comment form at the back of this book.

- · If you prefer to send comments by mail, use the readers' comment form with the address that is printed on the back. If you are mailing a readers' comment form from a country other than the United States, you can give the form to the local IBM branch office or IBM representative for postage-paid mailing.
- If you prefer to send comments by FAX, use either of the following numbers:
 - United States and Canada: 1-800-937-3430
 - Other countries: 1-507-253-5192
- If you prefer to send comments electronically, use one of these e-mail addresses:
 - Comments on books:

RCHCLERK@us.ibm.com

IBMMAIL, to IBMMAIL(USIB56RZ)

Comments on the iSeries Information Center:

RCHINFOC@us.ibm.com

Be sure to include the following:

- The name of the book.
- The publication number of the book.
- The page number or topic to which your comment applies.

Part 1. Run-Time Library Functions

Chapter 1. Include Files

The include files that are provided with the run time library contain macro and constant definitions, type definitions, and function declarations. Some functions require definitions and declarations from include files to work properly. The inclusion of files is optional, as long as the necessary statements from the files are coded directly into the source.

This section describes each include file, explains its contents, and lists the functions that are declared in the file.

The QSYSINC (system openness includes) library must be installed on your iSeries system. QSYSINC contains include files useful for C users, such as system API, Dynamic Screen Manager (DSM), and LE header files. The QSYSINC library contains header files that include the prototypes and templates for the MI built-ins and the ILE C MI functions. See the *ILE C/C++ for AS/400 MI Library Reference* for more information about these header files.

<assert.h>

The <assert.h> include file defines the assert macro. You must include assert.h when you use assert.

The definition of assert is in an #ifndef preprocessor block. If you have not defined the identifier NDEBUG through a #define directive or on the compilation command, the assert macro tests the assertion expression. If the assertion is false, the system prints a message to stderr, and raises an abort signal for the program. The system also does a Dump Job (DMPJOB) OUTPUT(*PRINT) when the assertion is false.

If NDEBUG is defined, assert is defined to do nothing. You can suppress program assertions by defining NDEBUG.

<ctype.h>

The <ctype.h> include file defines functions that are used in character classification. The functions that are defined in <ctype.h> are:

isascii ¹	isblank ²	isgraph	ispunct	$toascii^1$
isalnum	iscntrl	islower	isspace	tolower
isalpha	isdigit	isprint	isupper	toupper
			isxdiait	

Note: ¹ These functions are available with SYSIFCOPT(*IFSIO) AND LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) are specified on the compilation command.

Note: ² This function is applicable to C++ only.

<decimal.h>

The <decimal.h> include file contains definitions of constants that specify the ranges of the packed decimal type and its attributes. The <decimal.h> file must be included with a #include directive in your source code if you use the keywords decimal, digits of, or precision of.

<errno.h>

The <errno.h> include file defines macros that are set to the errno variable. The <errno.h> include file defines macros for values that are used for error reporting in the C library functions and defines the macro errno. An integer value can be assigned to errno, and its value can be tested during run time. See "Checking the Errno Value" in the WebSphere Development Studio: ILE C/C++ Programmer's Guide for information about displaying the current errno value.

Note: To test the value of errno after library function calls, set it to 0 before the call because its value may not be reset during the call.

<except.h>

The <except.h> include file declares types and macros that are used in ILE C exception handling.

The definition of _INTRPT_Hndlr_Parms_T is:

```
typedef Packed struct {
                  Block Size;
  unsigned int
  _INVFLAGS_T
                  Tgt Flags;
 char
                  reserved[8];
  INVPTR
                  Target;
 _INVPTR
                  Source;
  SPCPTR
                  Com_Area;
                  Compare_Data[32];
  char
  char
                  Msg Id[7];
  char
                  reserved1;
  INTRPT Mask T
                  Mask;
                  Msg_Ref_Key;
 unsigned int
 unsigned short Exception Id;
 unsigned short Compare Data Len;
                  Signal_Class;
 char
                  Priority;
 char
                  Severity;
 short
                  reserved3[4];
 char
  int
                  Msg Data Len;
                  Mch_Dep_Data[10];
 char
                  Tgt Inv Type;
 char
  SUSPENDPTR
                  Tgt Suspend;
                  Ex Data[48];
} INTRPT Hndlr Parms T;
```

Element

Description

Block Size

The size of the parameter block passed to the exception handler.

Tgt_Flags

Contains flags that are used by the system.

reserved

An eight byte reserved field.

Target An invocation pointer to the call stack entry that enabled the exception handler.

Source

An invocation pointer to the call stack entry that caused the exception. If that call stack entry no longer exists, then this is a pointer to the call stack entry where control resumes when the exception is handled.

Com_Area

A pointer to the communications area variable specified as the second parameter on the #pragma exception_handler. If a communication area was not specified, this value is NULL.

Compare_Data

The compare data consists of 4 bytes of message prefix, for example CPF, MCH, followed by 28 bytes which are taken from the message data of the related message. In the case where the message data is greater than 28 these are the first 28 bytes. For MCH messages, these are the first 28 bytes of the exception related data that is returned by the system (substitution text).

Msg_Id

A message identifier, for example CPF123D. *STATUS message types are not updated in this field.

reserved1

A 1 byte pad.

Mask This is an 8-byte exception mask, identifying the type of the exception that occurred, for example a decimal data error. The possible types are shown in Table 20 on page 461.

Msg_Ref_Key

A key used to uniquely identify the message.

Exception_Id

Binary value of the exception id, for example, 0x123D. To display value, use conversion specifier %x as information is stored in hex value.

Compare_Data_Len

The length of the compare data.

Signal_Class

Internal signal class.

Priority

The handler priority.

Severity

The message severity.

reserved3

A four-byte reserved field.

Msg_Data_Len

The length of available message data.

Mch Dep Data

Machine-dependent data.

Tgt_Inv_Type

Invocation type. Macros are defined in <mimchobs.h>.

Tgt_Suspend

Suspend pointer of the target.

Ex_Data

The first 48 bytes of exception data.

Element

Description

Block Size

The size of the parameter block passed to the cancel handler.

Inv_Flags

Contains flags that are used by the system.

reserved

An eight byte reserved field.

Invocation

An invocation pointer to the invocation that is being cancelled.

Com Area

A pointer to the handler communications area defined by the cancel handler.

Mask A 4 byte value indicating the cancel reason.

The following built-ins are defined in <except.h>:

Built-in

Description

__EXBDY

The purpose of the _EXBDY built-in or _EXBDY macro is to act as a boundary for exception-sensitive operations. An exception-sensitive operation is one that may signal an exception. An EXBDY enables programmers to selectively suppress optimizations that do code motion. For example, a divide is an exception-sensitive operation because it can signal a divide-by-zero. An execution path containing both an EXBDY and a divide will perform the two in the same order with or without optimization. For example:

```
b = exp1;
c = exp2;
...
_EXBDY();
a = b/c;
```

__VBDY

The purpose of a __VBDY built-in or _VBDY macro is to ensure the home storage locations are current for variables that are potentially used on exception paths. This ensures the visibility of the current values of variables in exception handlers. A VBDY enables programmers to selectively suppress optimizations, such as redundant store elimination and

forward store motion to enforce sequential consistency of variable updates. In the following example, the VBDYs ensure that state is in it's home storage location before each block of code that may signal an exception. A VBDY is often used in combination with an EXBDY to ensure that earlier assignments to state variables really update home storage locations and that later exception sensitive operations are not moved before these assignments.

```
state = 1;
 VBDY();
 7* Do stuff that may signal an exception.
                                                     */
 state = 2;
 VBDY();
 7* More stuff that may signal an exception.
state = 3;
_VBDY();
```

For more information on built-ins, see the ILE C/C++ for AS/400 MI Library Reference.

<float.h>

The <float.h> include file defines constants that specify the ranges of floating-point data types. For example, the maximum number of digits for objects of type double or the minimum exponent for objects of type float.

<langinfo.h>

The <langinfo.h> include file contains the declarations and definitions that are used by nl_langinfo.

limits.h>

The <1 imits.h> include file defines constants that specify the ranges of integer and character data types. For example, the maximum value for an object of type char.

<locale.h>

ı

The <locale.h> include file declares the setlocale() and localeconv() library functions. These functions are useful for changing the C locale when you are creating applications for international markets.

The <locale.h> include file also declares the type struct lconv and the following macro definitions:

```
NULL
                LC_ALL
                                LC C 1
                                                 LC_C_FRANCE LC_UCS2_ALL 3
LC_C_GERMANY LC_C_ITALY 1
                                LC_C_SPAIN 1
                                                 LC_C_UK 1
                                                                 LC_UCS2_CTYPE<sup>3</sup>
LC_C_USA 1
                                                 LC_MESSAGES <sup>2</sup> LC_UCS2_COLLATE<sup>3</sup>
                LC_COLLATE
                                LC_CTYPE
LC_TOD
                LC_MONETARY LC_NUMERIC
                                                 LC_TIME
```

Note: ¹ This macro is defined only when LOCALETYPE(*CLD) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

<math.h>

The <math.h> include file declares all the floating-point math functions:

acos	cos	floor	log	sqrt
asin	cosh	fmod	log10	tan
atan	erf	frexp	modf	tanh
atan2	erfc	gamma	pow	
Bessel	exp	hypot	sin	
ceil	fabs	1dexp	sinh	

Notes:

1.

Note: The Bessel functions are a group of functions named j0, j1, jn, y0, y1, and yn.

2.

Note: Floating point numbers are only guaranteed 15 significant digits. This can greatly affect expected results if multiple floating point numbers are used in a calculation.

<math.h> defines the macro HUGE VAL, which expands to a positive double expression, and possibly to infinity on systems that support infinity.

For all mathematical functions, a **domain error** occurs when an input argument is outside the range of values that are allowed for that function. In the event of a domain error, errno is set to the value of EDOM.

A range error occurs if the result of the function cannot be represented in a double value. If the magnitude of the result is too large (overflow), the function returns the positive or negative value of the macro HUGE_VAL, and sets errno to ERANGE. If the result is too small (underflow), the function returns zero.

<mallocinfo.h>

Include file with _C_TS_malloc_info and _C_TS_malloc_debug.

<monetary.h>

The <monetary.h> header file contains declarations and definitions that are related to the output of monetary quantities. These declarations and definitions are accessible only when LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

² This macro is defined only when LOCALETYPE(*LOCALE) is specified on the compilation command.

³ This macro is defined only when LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

<nl_types.h>

The <nl_types.h> header file contains catalog definitions. These definitions are accessible only when SYSIFCOPT(*IFSIO) with either LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

<pointer.h>

The <pointer.h> include file contains typedefs and pragma directives for the iSeries pointer types: space pointer, open pointer, invocation pointer, label pointer, system pointer, and suspend pointer. The typedefs _ANYPTR and _SPCPTRCN are also defined in <pointer.h>.

<recio.h>

The <recio.h> include file defines the types and macros, and prototypes functions for all the ILE C record input and output (I/O) operations.

The following functions are defined in <recio.h>:

_Rreadn

_Rupdate

_Rwrread

_Rreadl

_Rreads

_Rwriterd

_Racquire	_Rclose	_Rcommit	_Rdelete
_Rdevatr	_Rfeod	_Rfeov	_Rformat
_Rindara	_Riofbk	_Rlocate	_Ropen
_Ropnfbk	_Rpgmdev	_Rreadd	_Rreadf
_Rreadindv	_Rreadk	_Rreadl	_Rreadn
_Rreadnc	_Rreadp	_Rreads	_Rrelease
_Rrlslck	_Rrollbck	_Rupdate	_Rupfb
_Rwrite	_Rwrited	_Rwriterd	_Rwrread
The following pos	itioning macros are de	fined in recio.h:	
_END	END_FRC	FIRST	KEY_EQ
 KEY_GE	KEY_GT	 KEY_LE	CEY_LT
KEY_NEXTEQ	KEY_NEXTUNQ	KEY_PREVEQ	KEY_PREVUNQ
KEY_LAST	KEY_NEXT	NO_POSITION	PREVIOUS
PRIOR	RRN_EQ	START	START_FRC
LAST	NEXT		
The following made	cros are defined in rec	io.h:	
DATA_ONLY	DFT	NO_LOCK	NULL_KEY_MAP
The following dire	ectional macros are def	ined in recio.h:	
READ_NEXT	READ_PREV		
The following fun	ctions and macros sup	nort locate or move	mode:
The following full	chorb and macros sup	port locate of move.	iiioac.
_Rreadd	_Rreadf	_Rreadindv	_Rreadk

_Rreadnc

_Rwrite

_Rreadp

_Rwrited

Any of the record I/O functions that include a buffer parameter may work in move mode or locate mode. In move mode, data is moved between the user user-supplied buffer and the system buffer. In locate mode, the user must access the data in the system buffer. Pointers to the system buffers are exposed in the _RFILE structure. To specify that locate mode is being used, the buffer parameter of the record I/O function is coded as NULL.

A number of the functions include a size parameter. For move mode, this is the number of data bytes that are copied between the user-supplied buffer and the system buffer. All of the record I/O functions work with one record at a time regardless of the size that is specified. The size of this record is defined by the file description. It may not be equal to the size parameter that is specified by the user on the call to the record I/O functions. The amount of data that is moved between buffers is equal to the record length of the current record format or specified minimum size, whichever is smaller. The size parameter is ignored for locate mode.

The following types are defined in recio.h:

Information for controlling opened record I/O operations

```
typedef Packed struct {
   char
                                                  reserved1[16];
   volatile void *const *const in_buf;
   volatile void *const *const out buf;
 couz[48];
riofb;
reserved3[32];
buf_length;
reserved4[28];
volatile char *const in_null_map;
volatile char *const out_null_map;
volatile char *const null_key_man*
char
const int
short
short
   char
                                                 reserved2[48];
                                                 null_key_map_len;
   short
                                                  reserved6[8];
   char
} RFILE;
```

Element

Description

in_null_map

Specifies which fields are to be considered NULL when you read from a database file.

out_null_map

Specifies which fields are to be considered NULL when you write to a database file.

null_key_map

Specifies which fields contain NULL if you are reading a database by key.

null_map_len

Specifies the lengths of the in_null_map and out_null_map.

null_key_map_len

Specifies the length of the null_key_map.

Record I/O Feedback Information

```
typedef struct {
 unsigned char
                   *key;
  Sys Struct T
                   *sysparm;
 unsigned long
                   rrn;
                   num bytes;
 long
 short
                   blk count;
  char
                   blk filled by;
                   dup key
 int
                   icf_locate :1;
 int
  int
                   reserved1 :6;
  char
                   reserved2[20];
} RIOFB T;
```

Element

Description

key If you are processing a file using a keyed sequence access path, this field contains a pointer to the key value of the record successfully positioned to, read or written.

sysparm

This field is a pointer to the major and minor return code for ICF, display, and printer files.

This field contains the relative record number of the record that was rrn successfully positioned to, read or written.

num_bytes

This field contains the number of bytes that are read or are written.

blk count

This field contains the number of records that remain in the block. If the file is open for input, blkrcd=y is specified, and a read function is called, this field will be updated with the number of records remaining in the block.

blk filled by

This field indicates the operation that filled the block. If the file is open for input, blkrcd=y is specified, and a read function is called. This field will be set to the __READ_NEXT macro if the _Rreadn function filled the block or to the __READ_PREV macro if the _Rreadp function filled the block.

System—Specific Information

```
typedef struct {
                    *sysparm ext;
 void
  _Maj_Min_rc_T
                   Maj Min;
 char
                    reserved1[12];
} Sys Struct T;
```

Major and Minor Return Codes

```
typedef struct {
           major rc[2];
     char
      char
             minor rc[2];
} _Maj_Min_rc_T;
```

The following macros are defined in recio.h:

_FILENAME_MAX

Expands to an integral constant expression that is the size of a character array large enough to hold the longest file name. This is the same as the stream I/O macro.

_ROPEN_MAX

Expands to an integral constant expression that is the maximum number of files that can be opened simultaneously.

The following null field macros are defined in recio.h:

Element

Description

_CLEAR_NULL_MAP(file, type)

Clears the null output field map that indicates that there are no null fields in the record to be written to file. type is a typedef that corresponds to the null field map for the current record format.

_CLEAR_UPDATE_NULL_MAP(file, type)

Clears the null input field map that indicates that no null fields are in the record to be written to file. type is a typedef that corresponds to the null field map for the current record format.

_QRY_NULL_MAP(file, type)

Returns the number of fields that are null in the previously read record. type is a typedef that corresponds to the null field map for the current record format.

CLEAR NULL KEY MAP(file, type)

Clears the null key field map so that it indicates no null key fields in the record to be written to file. type is a typedef that corresponds to the null key field map for the current record format.

_SET_NULL_MAP_FIELD(file, type, field)

Sets the specified field in the output null field map so that *field* is considered NULL when the record is written to file.

_SET_UPDATE_NULL_MAP_FIELD(file, type, field)

Sets the specified field in the input null field map so that field is considered null when the record is written to file. type is a typedef that corresponds to the null key field map for the record format.

_QRY_NULL_MAP_FIELD(file, type, field)

Returns 1 if the specified field in the null input field map indicates that the field is to be considered null in the previously read record. If field is not null, it returns zero. type is a typedef that corresponds to the NULL key field map for the current record format.

_SET_NULL_KEY_MAP_FIELD(file, type, field)

Sets the specified field map that indicates that the field will be considered null when the record is read from file. type is a typedef that corresponds to the null key field map for the current record format.

_QRY_NULL_KEY_MAP(file, type)

Returns the number of fields that are null in the key of the previously read record. type is a typedef that corresponds to the null field map for the current record format.

QRY NULL KEY MAP FIELD(file, type, field)

Returns 1 if the specified field in the null key field map indicates that field is to be considered null in the previously read record. If field is not null, it returns zero. type is a typedef that corresponds to the null key field map for the current record format.

<regex.h>

The <regex.h> include file defines the following regular expression functions:

regcomp() regerror() regexec() regfree()

The <regex.h> include file also declares the regmatch_t type, the regex_t type, which is capable of storing a compiled regular expression, and the following macros:

Values of the *cflags* parameter of the regcomp() function:

REG BASIC

REG_EXTENDED

REG_ICASE

REG_NEWLINE

REG_NOSUB

Values of the *eflags* parameter of the regexec() function:

REG_NOTBOL

REG_NOTEOL

Values of the *errcode* parameter of the regerror() function:

REG NOMATCH

REG_BADPAT

REG_ECOLLATE

REG ECTYPE

REG EESCAPE

REG_ESUBREG

REG_EBRACK

REG_EPAREN

REG_EBRACE

REG BADBR

REG_ERANGE

REG ESPACE

REG_BADRPT

REG_ECHAR

REG EBOL

REG_EEOL

REG_ECOMP

REG_EEXEC REG_LAST

These declarations and definitions are accessible only when LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

<setjmp.h>

The <setjmp.h> include file declares the setjmp() macro and longjmp function. It also defines a buffer type, jmp_buf, that the setjmp() macro and longjmp function use to save and restore the program state.

<signal.h>

The <signal.h> include file defines the values for signals and declares the signal() and raise() functions.

The <signal.h> include file also defines the following macros:

SIGABRT	SIG_ERR	SIGILL	SIGOTHER	SIGUSR1
SIGALL	SIGFPE	SIGINT	SIGSEGV	SIGUSR2
SIG_DFL	SIG_IGN	SIGIO	SIGTERM	

<signal.h> also declares the function _GetExcData, an iSeries extension to the C standard library.

<stdarg.h>

The <stdarg.h> include file defines macros that allow you access to arguments in functions with variable-length argument lists: va_arg(), va_start(), and va_end(). The <stdarg.h> include file also defines the type va_list.

<stddef.h>

The <stddef.h> include file declares the commonly used pointers, variables, and types as listed below:

ptrdiff t

typedef for the type of the difference of two pointers

size_t typedef for the type of the value that is returned by sizeof

wchar t

typedef for a wide character constant.

The <stddef.h> include file also defines the macros NULL and offsetof. NULL is a pointer that is guaranteed not to point to a data object. The offsetof macro expands to the number of bytes between a structure member and the start of the structure. The offsetof macro has the form:

```
offsetof(structure_type, member)
```

The <stddef.h> include file also declares the extern variable _EXCP_MSGID, an iSeries extension to C.

<stdio.h>

The <stdio.h> include file defines constants, macros, and types, and declares stream input and output functions. The stream I/O functions are:

clearerr fclose feof ferror fflush fgetc fgetpos fgets fgetwc ¹ fgetws ¹ fileno² fopen fprintf fputc _fputchar	fputs fputwc¹ fputws¹ fread freopen fscanf fseek fsetpos ftell fwide ¹ fwprintf ¹ fwrite fwscanf ¹ getc	getchar gets getwc 1 getwchar 1 perror printf putc putchar puts putwc 1 putwchar 1 remove rename rewind	scanf setbuf setvbuf snprintf sprintf sscanf tmpfile tmpnam ungetc ungetwc ¹ vfprintf vfwprintf ¹ vprintf vsnprintf vsnprintf vsprintf	wprintf ¹ wscanf ¹
_iputchar			vwprintf ¹	

Note:

The <stdio.h> include file also defines the macros that are listed below. You can use these constants in your programs, but you should not alter their values.

BUFSIZ Specifies the buffer size that the setbuf library function will use when you are allocating buffers for stream I/O. This value establishes the size of system-allocated buffers and is used with setbuf.

E0F The value that is returned by an I/O function when the end of the file (or in some cases, an error) is found.

FOPEN MAX

The number of files that can be opened simultaneously.

FILENAME_MAX

The longest file name that is supported. If there is no reasonable limit, FILENAME_MAX will be the recommended size.

L_tmpnam

The size of the longest temporary name that can be generated by the tmpnam function.

TMP MAX

The minimum number of unique file names that can be generated by the tmpnam function.

NULL A pointer guaranteed not to point to a data object.

The FILE structure type is defined in <stdio.h>. Stream I/O functions use a pointer to the FILE type to get access to a given stream. The system uses the information in the FILE structure to maintain the stream.

¹ These functions are available when SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) are specified on the compilation command.

² These functions are available when SYSIFCOPT(*IFSIO)is specified on the compilation command.

When integrated file system is enabled with a compilation parameter SYSIFCOPT(*IFSIO), ifs.h is included into <stdio.h>.

The C standard streams stdin, stdout, and stderr are also defined in <stdio.h>.

The macros SEEK_CUR, SEEK_END, and SEEK_SET expand to integral constant expressions and can be used as the third argument to fseek().

The macros _IOFBF, _IOLBF, and _IONBF expand to integral constant expressions with distinct values suitable for use as the third argument to the setvbuf function.

The type fpos_t is defined in <stdio.h> for use with fgetpos() and fsetpos().

See "<stddef.h>" on page 14 for more information on NULL.

<stdlib.h>

The <stdlib.h> include file declares the following functions:

abort abs atexit atof atoi atol	bsearch calloc div exit free _gcvt ¹ getenv _itoa ¹ _ltoa ¹	ldiv lldiv malloc mblen mbstowcs mbtowc putenv	qsort rand rand_r realloc srand strtod	strtol strtoul strtoull system _ultoa ¹ wctomb wcstombs
	llabs			

Note: ¹ These functions are applicable to C++ only.

The <stdlib.h> include file also contains definitions for the following macros:

NULL The NULL pointer value.

EXIT_SUCCESS

Expands to 0; used by the atexit function.

EXIT FAILURE

Expands to 8; used by the atexit function.

RAND MAX

Expands to an integer that represents the largest number that the rand function can return.

MB CUR MAX

Expands to an integral expression to represent the maximum number of bytes in a multibyte character for the current locale.

For more information on NULL and the types size_t and wchar_t, see "<stddef.h>" on page 14.

<string.h>

The <string.h> include file declares the string manipulation functions:

memchr	strcat	strcspn	strncmp	$strset^1$
memcmp	strchr	$strdup^1$	strncpy	strspn
memcpy	strcmp	strerror	$strnicmp^1$	strstr
$memicmp^1$	${\sf strcmpi}^1$	$stricmp^1$	$strnset^1$	strtok
memmove	strcoll	strlen	strpbrk	strtok_r
memset	strcpy	strncat	strrchr	strxfrm

Note: ¹ These functions are applicable to C++ only.

The <string.h> include file also defines the macro NULL, and the type size_t.

For more information on NULL and the type size_t, see "<stddef.h>" on page 14.

<strings.h>

Contains the functions strcasecmp and strncasecmp

<time.h>

The <time.h> include file declares the time and date functions:

asctime	ctime	gmtimegmtime_r	localtime_r	time
asctime_r	ctime_r	localtime	mktime	
clock	difftime		strftime	

The <time.h> include file also provides:

- A structure tm that contains the components of a calendar time. See "gmtime() — Convert Time" on page 141 for a list of the tm structure members.
- A macro CLOCKS PER SEC equal to the number per second of the value that is returned by the clock function.
- Types clock t, time t, and size t.
- The NULL pointer value.

For more information on NULL and the type size_t, see "<stddef.h>" on page 14.

<wchar.h>

The <wchar.h> header file contains declarations and definitions that are related to the manipulation of wide character strings. Any functions which deal with files are accessible if SYSIFCOPT(*IFSIO) is specified.

btowc ¹	${\sf mbsinit}^1$	vswprintf ¹	$wcsrtombs^1$	$wcwidth^1$
fgetwc ²	mbrlen¹	vwprintf ²	wcsstr	wmemchr
fgetws ²	mbrtowc ¹	$wcrtomb^1$	$wcstod^1$	wmemcmp
fputwc ²	mbsrtowcs ¹	wcsicmp ¹	wcstok	wmemcpy
fputws ²	putwc²	wcsnicmp ¹	wcstol ¹	wmemmove
fwide ²	putwchar ²	wcscoll ¹	wcstoul ¹	wmemset
fwprintf ²	$swprintf^1$	wcstoll ¹	$wcswidth^1$	
fwscanf ²	swscanf ²	wcstoull ¹	$\mathrm{wcsxfrm}^1$	
getwc ²	ungetwc²	wcscspn	$wctob^1$	
getwchar ²	vfwprintf ²	$wcsftime^1$		
		wscanf ²		
		wprintf ²		

Note:

<wcstr.h>

The<wcstr.h> include file declares the multibyte functions:

wcscat	wcscpy	wcsncat	wcspbrk	WCSWCS
wcschr	wcscspn	wcsncmp	wcsspn	
wcscmp	wcslen	wcsncpy	wcsrchr	

<wcstr.h> also defines the types size_t, NULL, and wchar_t.

For more information on NULL and the types size_t and wchar_t, see "<stddef.h>" on page 14.

<wctype.h>

The <wctype.h> header file contains declarations and definitions for wide character classification. These declarations and definitions are accessible only when LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

<xxcvt.h>

The <xxcvt.h> include file contains the declarations that are used by the QXXDTOP, QXXDTOZ, QXXITOP, QXXITOZ, QXXPTOI, QXXPTOD, QXXZTOD, and QXXZTOI conversion functions.

¹ These functions are available only when LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

² These functions are available only when SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALE) are specified on the compilation command.

<xxdtaa.h>

The <xxdtaa.h> include file contains the declarations for the data area interface functions QXXCHGDA, QXXRTVDA, and the type _DTAA_NAME_T.

```
The definition of _DTAA_NAME_T is:
typedef struct DTAA NAME T {
       char dtaa name[10];
       char dtaa_lib[10];
}_DTAA_NAME_T;
```

<xxenv.h>

The <xxenv.h> include file contains the declarations for the QPXXCALL and QPXXDLTE EPM environment handling program. ILE procedures cannot be called from this interface.

```
The definition of _ENVPGM_T is:
typedef struct _ENVPGM_T {
    char pgmname[10];
    char pgmlib[10];
} _ENVPGM_T;
```

<xxfdbk.h>

The <xxfdbk.h> include file contains the declarations that are used by the OS/400 system feedback areas. To retrieve information from feedback areas, see "_Riofbk() —Obtain I/O Feedback Information" on page 252 and "_Ropnfbk() — Obtain Open Feedback Information" on page 261.

The following is an example of a type that is defined in the <xxfdbk.h> include

```
typedef _Packed struct _XXIOFB_T {
    short
                file_dep_fb_offset;
                write count;
    int
    int
                read count;
                write read count;
    int
                other_io_count;
    int
                reserved1;
    char
    char
                cur operation;
    char
                rec format[10];
                dev_class[2];
    char
                dev_name[10];
    char
                last_io_rec_len;
    int
    char
                reserved2[80];
    short
                num recs retrieved;
    short
                last io rec len2;
    char
                reserved3[2];
                cur blk count;
    int
    char
                reserved4[8];
} _XXIOFB_T;
```

For further information on the open feedback areas, see the File Management topic in the Information Center.

Machine Interface (MI) Include Files

See the ILE C/C++ for AS/400 MI Library Reference for a description of the MI header files.

Chapter 2. Library Functions

This chapter describes the standard C library functions and the ILE C extensions to the library functions, except for the ILE C MI functions. See the *ILE C/C++ for AS/400 MI Library Reference* for more information about the MI functions.

Each library function that is listed in this section contains:

- A format description that shows the include file that declares the function.
- The data type that is returned by the function.
- The required data types of the arguments to the function.

This example shows the format of the log() function:

```
#include <math.h>
double log(double x);
```

The example shows that:

- you must include the file math.h in the program.
- the log() function returns type double.
- the log() function requires an argument *x* of type double.

Examples throughout the section illustrate the use of library functions and are not necessarily complete.

This chapter lists the library functions in alphabetic order. If you are unsure of the function you want to use, see the summary of the library functions in "The C Library".

The C Library

This chapter summarizes the available C library functions and their location in this book. It also briefly describes what the function does. Each library function is listed according to the type of function it performs.

Error Handling

Function	Header File	Page	Description
assert()	assert.h	44	Prints diagnostic messages.
atexit()	stdlib.h	46	Registers a function to be executed at program end.
clearerr()	stdio.h	62	Resets error indicators.
feof()	stdio.h	79	Tests end-of-file indicator for stream input.
ferror()	stdio.h	79	Tests the error indicator for a specified stream.
perror()	stdio.h	198	Prints an error message to stderr.

Function	Header File	Page	Description
_GetExcData()	signal.h	"_GetExcData() — Get Exception Data" on page 135	Retrieves information about an exception from within a C signal handler. This function is not defined when SYSIFCOPT(*SYNCSIGNAL) is specified on the compilation command.
raise()	signal.h	226	Initiates a signal.
signal()	signal.h	313	Allows handling of an interrupt signal from the operating system.
strerror()	string.h	333	Retrieves pointer to system error message.

Searching and Sorting

Function	Header File	Page	Description
bsearch()	stdlib.h	52	Performs a binary search of a sorted array.
qsort()	stdlib.h	216	Performs a quick sort on an array of elements.

Mathematical

Function	Header File	Page	Description
abs()	stdlib.h	38	Calculates the absolute value of an integer.
ceil()	math.h	61	Calculates the double value representing the smallest integer that is greater than or equal to a number.
div()	stdlib.h	70	Calculates the quotient and remainder of an integer.
erf()	math.h	72	Calculates the error function.
erfc()	math.h	72	Calculates the error function for large numbers.
exp()	math.h	73	Calculates an exponential function.
fabs()	math.h	74	Calculates the absolute value of a floating-point number.
floor()	math.h	91	Calculates the double value representing the largest integer that is less than or equal to a number.
fmod()	math.h	91	Calculates the floating point remainder of one argument divided by another.
frexp()	math.h	112	Separates a floating-point number into its mantissa and exponent.
gamma()	math.h	131	Calculates the gamma function.

Function	Header File	Page	Description
hypot()	math.h	145	Calculates the hypotenuse.
labs()	stdlib.h	155	Calculates the absolute value of a long integer.
11abs()	stdlib.h	155	Calculates the absolute value of a long long integer.
ldexp()	math.h	156	Multiplies a floating-point number by an integral power of 2.
ldiv()	stdlib.h	156	Calculates the quotient and remainder of a long integer.
lldiv()	stdlib.h	156	Calculates the quotient and remainder of a long long integer.
log()	math.h	165	Calculates natural logarithm.
log10()	math.h	166	Calculates base 10 logarithm.
modf()	math.h	195	Calculates the signed fractional portion of the argument.
pow()	math.h	199	Calculates the value of an argument raised to a power.
sqrt()	math.h	318	Calculates the square root of a number.

Trigonometric Functions

Function	Header File	Page	Description
acos()	math.h	39	Calculates the arc cosine.
asin()	math.h	43	Calculates the arc sine.
atan()	math.h	45	Calculates the arc tangent.
atan2()	math.h	45	Calculates the arc tangent.
cos()	math.h	64	Calculates the cosine.
cosh()	math.h	65	Calculates the hyperbolic cosine.
sin()	math.h	315	Calculates the sine.
sinh()	math.h	316	Calculates the hyperbolic sine.
tan()	math.h	371	Calculates the tangent.
tanh()	math.h	372	Calculates the hyperbolic tangent.

Bessel Functions

Function	Header File	Page	Description
j0()	math.h	51	0 order differential equation of the first kind.
j1()	math.h	51	1st order differential equation of the first kind.
jn()	math.h	51	<i>n</i> th order differential equation of the first kind.
y0()	math.h	51	0 order differential equation of the second kind.

Function	Header File	Page	Description
y1()	math.h	51	1st order differential equation of the second kind.
yn()	math.h	51	<i>n</i> th order differential equation of the second kind.

Time Manipulation

Function	Header File	Page	Description
asctime()	time.h	40	Converts time stored as a structure to a character string in storage.
asctime_r()	time.h	42	Converts time stored as a structure to a character string in storage. (Restartable version of asctime())
clock()	time.h	63	Determines processor time.
ctime()	time.h	66	Converts time stored as a long value to a character string.
ctime_r()	time.h	67	Converts time stored as a long value to a character string. (Restartable version of ctime())
difftime()	time.h	69	Calculates the difference between two times.
gmtime()	time.h	141	Converts time to Coordinated Universal Time structure.
gmtime_r()	time.h	143	Converts time to Coordinated Universal Time structure. (Restartable version of gmtime())
localtime()	time.h	163	Converts time to local time.
localtime_r()	time.h	164	Converts time to local time. (Restartable version of localtime())
mktime()	time.h	193	Converts local time into calendar time.
time()	time.h	373	Returns the time in seconds.

Type Conversion

Function	Header File	Page	Description
atof()	stdlib.h	47	Converts a character string to a floating-point value.
atoi()	stdlib.h	49	Converts a character string to an integer.
atol()	stdlib.h	50	Converts a character string to a long integer.
strtod()	stdlib.h	357	Converts a character string to a double value.
strtol()	stdlib.h	362	Converts a character string to a long integer.

Function	Header File	Page	Description
strtoll()	stdlib.h	362	Converts a character string to a long long integer.
strtoul()	stdlib.h	364	Converts a string to an unsigned long integer.
strtoull()	stdlib.h	364	Converts a string to an unsigned long long integer.
toascii()	ctype.h	376	Converts a character to the corresponding ASCII value.
wcstod()	wchar.h	422	Converts a wide-character string to a double floating point.
wcstol()	wchar.h	424	Converts a wide-character string to a long integer.
wcstoll	wchar.h	424	Converts a wide-character string to a long long integer.
wcstoul()	wchar.h	430	Converts a wide-character string to an unsigned long integer.
wcstuoll	wchar.h	430	Converts a wide-character string to an unsigned long long integer.

Conversion

Function	Header File	Page	Description
QXXDTOP()	xxcvt.h	219	Converts a floating-point value to a packed decimal value.
QXXDTOZ()	xxcvt.h	220	Converts a floating-point value to a zoned decimal value.
QXXITOP()	xxcvt.h	221	Converts an integer value to a packed decimal value.
QXXITOZ()	xxcvt.h	221	Converts an integer value to a zoned decimal value.
QXXPTOD()	xxcvt.h	222	Converts a packed decimal value to a floating-point value.
QXXPTOI()	xxcvt.h	223	Converts a packed decimal value to an integer value.
QXXZTOD()	xxcvt.h	224	Converts a zoned decimal value to a floating-point value.
QXXZTOI()	xxcvt.h	225	Converts a zoned decimal value to an integer value.

Record Input/Output

Function	Header File	Page	Description
_Racquire()	recio.h	228	Prepares a device for record I/O operations.
_Rclose()	recio.h	229	Closes a file that is opened for record I/O operations.

Function	Header File	Page	Description
_Rcommit()	recio.h	230	Completes the current transaction, and establishes a new commitment boundary.
_Rdelete()	recio.h	232	Deletes the currently locked record.
_Rdevatr()	recio.h xxfdbk.h	233	Returns a pointer to a copy of the device attributes feedback area for the file reference by fp and the device pgmdev.
_Rfeod()	recio.h	246	Forces an end-of-file condition for the file referenced by fp.
_Rfeov()	recio.h	247	Forces an end-of-volume condition for tapes.
_Rformat()	recio.h	249	Sets the record format to fmt for the file referenced by fp.
_Rindara()	recio.h	251	Sets up the separate indicator area to be used for subsequent record I/O operations.
_Riofbk()	recio.h xxfdbk.h	252	Returns a pointer to a copy of the I/O feedback area for the file referenced by fp.
_Rlocate()	recio.h	254	Positions to the record in the files associated with fp and specified by the key, klen_rrn and opt parameters.
_Ropen()	recio.h	257	Opens a file for record I/O operations.
_Ropnfbk()	recio.h xxfdbk.h	261	Returns a pointer to a copy of the open feedback area for the file referenced by fp.
_Rpgmdev()	recio.h	262	Sets the default program device.
_Rreadd()	recio.h	263	Reads a record by relative record number.
_Rreadf()	recio.h	265	Reads the first record.
_Rreadindv()	recio.h	267	Reads data from an invited device.
_Rreadk()	recio.h	270	Reads a record by key.
_Rreadl()	recio.h	274	Reads the last record.
_Rreadn()	recio.h	275	Reads the next record.
_Rreadnc()	recio.h	278	Reads the next changed record in the subfile.
_Rreadp()	recio.h	279	Reads the previous record.
_Rreads()	recio.h	282	Reads the same record.
_Rrelease()	recio.h	283	Makes the specified device ineligible for record I/O operations.
_Rrlslck()	recio.h	285	Releases the currently locked record.
_Rrollbck()	recio.h	286	Reestablishes the last commitment boundary as the current commitment boundary.

I
I
I
I
I
I
I

Function	Header File	Page	Description
_Rupdate()	recio.h	288	Writes to the record that is currently locked for update.
_Rupfb()	recio.h	290	Updates the feedback structure with information about the last record I/O operation.
_Rwrite()	recio.h	291	Writes a record to the end of the file.
_Rwrited()	recio.h	293	Writes a record by relative record number. It will only write over deleted records.
_Rwriterd()	recio.h	296	Writes and reads a record.
_Rwrread()	recio.h	297	Functions as _Rwriterd(), except separate buffers may be specified for input and output data.

Stream Input/Output

Formatted Input/Output

Function	Header File	Page	Description
fprintf()	stdio.h	99	Formats and prints characters to the output stream.
fscanf()	stdio.h	113	Reads data from a stream into locations given by arguments.
fwprintf()	stdio.h	123	Format data as a wide character and write it to a stream.
fwscanf()	stdio.h	128	Read data from stream using wide-character string.
printf()	stdio.h	200	Formats and prints characters to stdout.
scanf()	stdio.h	299	Reads data from stdin into locations given by arguments.
snprintf()	stdio.h	"snprintf() — Print Formatted Data to Buffer" on page 320	Same as sprintf, except that the snprintf() function will stop after n characters have been written to outbuf.
sprintf()	stdio.h	317	Formats and writes characters to a buffer.

Function	Header File	Page	Description
sscanf()	stdio.h	322	Reads data from a buffer into locations given by arguments.
swprintf()	wchar.h	367	Format and write wide characters to buffer.
swscanf()	wchar.h	369	Reads wide data from a buffer into locations given by arguments.
vfprintf()	stdio.h stdarg.h	386	Formats and prints characters to the output stream using a variable number of arguments.
vfwprintf()	stdio.h stdarg.h	387	Format argument data as wide characters and write to a stream.
vprintf()	stdarg.h stdio.h	389	Formats and writes characters to stdout using a variable number of arguments.
vsnprintf()	stdio.h stdarg.h	390	Same as vsprintf, except that the vsnprintf function will stop after n characters have been written to outbuf.
vsprintf()	stdarg.h stdio.h	391	Formats and writes characters to a buffer using a variable number of arguments.
vswprintf()	stdio.h stdarg.h	393	Format and write wide characters to buffer using a variable number of arguments.
vwprintf()	stdio.h stdarg.h	394	Format argument data as wide characters and print using a variable number of arguments.
wprintf()	wchar.h	447	Format data as wide characters and print.

Function	Header File	Page	Description
wscanf()	wchar.h	448	Read data using wide-character format string.

Character and String Input/Output

Function	Header File	Page	Description
fgetc()	stdio.h	82	Reads a character from a specified input stream.
fgets()	stdio.h	85	Reads a string from a specified input stream.
fgetwc()	stdio.h	86	Reads wide character from stream.
fgetws()	stdio.h	88	Reads wide-character string from stream.
fputc()	stdio.h	101	Prints a character to a specified output stream.
fputs()	stdio.h	103	Prints a string to a specified output stream.
fputwc()	stdio.h	104	Writes wide character.
fputws()	stdio.h	106	Writes wide-character string.
getc()	stdio.h	133	Reads a character from a specified input stream.
getchar()	stdio.h	133	Reads a character from stdin.
gets()	stdio.h	137	Reads a line from stdin.
getwc()	stdio.h	138	Reads wide character from stream.
getwchar()	stdio.h	140	Gets wide character from stdin.
putc()	stdio.h	210	Prints a character to a specified output stream.
putchar()	stdio.h	210	Prints a character to stdout.
puts()	stdio.h	212	Prints a string to stdout.
putwc()	stdio.h	213	Writes wide character.
putwchar()	stdio.h	214	Writes wide character to stdout.
ungetc()	stdio.h	381	Pushes a character back onto a specified input stream.
ungetwc()	stdio.h	382	Pushes wide character onto input stream.

Direct Input/Output

Function	Header File	Page	Description
fread()	stdio.h	108	Reads items from a specified input stream.
fwrite()	stdio.h	127	Writes items to a specified output stream.

File Positioning

Function	Header File	Page	Description
fgetpos()	stdio.h	84	Gets the current position of the file pointer.
fseek()	stdio.h	114	Moves the file pointer to a new location.
fseeko()	stdio.h	"fseek() — fseeko() — Reposition File Position" on page 114	Same as fseek().
fsetpos()	stdio.h	117	Moves the file pointer to a new location.
ftell()	stdio.h	118	Gets the current position of the file pointer.
ftello()	stdio.h	118	Same as ftell().
rewind()	stdio.h	245	Repositions the file pointer to the beginning of the file.

File Access

Function	Header File	Page	Description
fclose()	stdio.h	75	Closes a specified stream.
fdopen()	stdio.h	76	Associates an input or output stream with a file.
fflush()	stdio.h	80	Causes the system to write the contents of a buffer to a file.
fopen()	stdio.h	92	Opens a specified stream.
freopen()	stdio.h	111	Closes a file and reassigns a stream.
fwide()	stdio.h	120	Determines stream orientation.
setbuf()	stdio.h	305	Allows control of buffering.
setvbuf()	stdio.h	311	Controls buffering and buffer size for a specified stream.
wfopen()	stdio.h	442	Opens a specified stream, accepting file name and mode as wide characters.

File Operations

Function	Header File	Page	Description
fileno()	stdio.h	90	Determines the file handle.
remove()	stdio.h	243	Deletes a specified file.
rename()	stdio.h	244	Renames a specified file.

Function	Header File	Page	Description
<pre>tmpfile()</pre>	stdio.h	374	Creates a temporary file and returns a pointer to that file.
tmpnam()	stdio.h	375	Produces a temporary file name.

Handling Argument Lists

Function	Header File	Page	Description
va_arg()	stdarg.h	384	Allows access to variable number of function arguments.
va_end()	stdarg.h	384	Allows access to variable number of function arguments.
va_start()	stdarg.h	384	Allows access to variable number of function arguments.

Pseudorandom Numbers

Function	Header File	Page	Description
rand(), rand_r()	stdlib.h	227	Returns a pseudorandom integer. (rand_r() is the restartable version of rand().)
srand()	stdlib.h	319	Sets the starting point for pseudorandom numbers.

Dynamic Memory Management

Function	Header File	Page	Description
_C_TS_malloc_info	mallocinfo.h	170	Returns the current memory usage information.
_C_TS_malloc_debug	mallocinfo.h	170	Returns the same information as _C_TS_malloc_info, plus produces a spool file of detailed information about the memory structure used by malloc functions when compiled with teraspace.
calloc()	stdlib.h	56	Reserves storage space for an array and initializes the values of all elements to 0.
free()	stdlib.h	109	Frees storage blocks.
malloc()	stdlib.h	170	Reserves storage blocks.
realloc()	stdlib.h	234	Changes storage size allocated for an object.

Memory Objects

Function	Header File	Page	Description
memchr()	string.h	187	Searches a buffer for the first occurrence of a given character.
memcmp()	string.h	188	Compares two buffers.
memcpy()	string.h	189	Copies a buffer.
memmove()	string.h	191	Moves a buffer.
memset()	string.h	192	Sets a buffer to a given value.
wmemchr()	wchar.h	442	Locates a wide character in a wide-character buffer.
wmemcmp()	wchar.h	443	Compares two wide-character buffers.
wmemcpy()	wchar.h	444	Copies a wide-character buffer.
wmemmove()	wchar.h	445	Moves a wide-character buffer.
wmemset()	wchar.h	446	Sets a wide-character buffer to a given value.

Environment Interaction

Function	Header File	Page	Description
abort()	stdlib.h	37	Ends a program abnormally.
exit()	stdlib.h	73	Ends the program normally if called in the initial thread.
getenv()	stdlib.h	135	Searches environment variables for a specified variable.
_C_Get_Ssn_Handle()	stdio.h	55	Returns a handle to the C session for use with DSM APIs.
localeconv()	locale.h	158	Formats numeric quantities in struct lconv according to the current locale.
longjmp()	setjmp.h	168	Restores a stack environment.
nl_langinfo()	langinfo.h	196	Retrieves information from the current locale.
putenv()	stdlib.h	211	Sets the value of an environment variable by altering an existing variable or creating a new one.
setjmp()	setjmp.h	307	Saves a stack environment.
setlocale()	locale.h	308	Changes or queries locale.

Function	Header File	Page	Description
system()	stdlib.h		Passes a string to the operating system's command interpreter.

String Operations

Function	Header File	Page	Description
strcasecmp	strings.h	"strcasecmp() — Compare Strings without Case Sensitivity" on page 323	Compares strings without case sensitivity.
strncasecmp	strings.h	"strncasecmp() — Compare Strings without Case Sensitivity" on page 341	Compares strings without case sensitivity.
strcat()	string.h	324	Concatenates two strings.
strchr()	string.h	325	Locates the first occurrence of a specified character in a string.
strcmp()	string.h	326	Compares the value of two strings.
strcoll()	string.h	329	Compares the locale-defined value of two strings.
strcpy()	string.h	330	Copies one string into another.
strcspn()	string.h	331	Finds the length of the first substring in a string of characters not in a second string.
strfmon()	string.h	334	Converts monetary value to string.
strftime()	time.h	336	Converts date and time to a formatted string.
strlen()	string.h	341	Calculates the length of a string.
strncat()	string.h	343	Adds a specified length of one string to another string.
strncmp()	string.h	344	Compares two strings up to a specified length.
strncpy()	string.h	346	Copies a specified length of one string into another.
strpbrk()	string.h	348	Locates specified characters in a string.
strptime()	time.h	350	Converts string to formatted time.
strrchr()	string.h	354	Locates the last occurrence of a character within a string.
strspn()	string.h	355	Locates the first character in a string that is not part of specified set of characters.
strstr()	string.h	356	Locates the first occurrence of a string in another string.

Function	Header File	Page	Description
strtok()	string.h	359	Locates a specified token in a string.
strtok_r()	string.h	361	Locates a specified token in a string. (Restartable version of strtok()).
strxfrm()	string.h	366	Transforms strings according to locale.
wcsftime()	wchar.h	407	Converts to formatted date and time.
wcsstr()	wchar.h	421	Locates a wide-character substring.
wcstok()	wchar.h	429	Tokenizes a wide-character string.

Character Testing

Function	Header File	Page	Description
isalnum()	ctype.h	147	Tests for alphanumeric characters.
isalpha()	ctype.h	147	Tests for alphabetic characters.
isascii()	ctype.h	146	Tests for ASCII values.
iscntrl()	ctype.h	147	Tests for control characters.
isdigit()	ctype.h	147	Tests for decimal digits.
isgraph()	ctype.h	147	Tests for printable characters excluding the space.
islower()	ctype.h	147	Tests for lowercase letters.
isprint()	ctype.h	147	Tests for printable characters including the space.
ispunct()	ctype.h	147	Tests for printable characters excluding the space.
isspace()	ctype.h	147	Tests for white-space characters.
isupper()	ctype.h	147	Tests for uppercase letters.
isxdigit()	ctype.h	147	Tests for wide hexadecimal digits 0 through 9, a through f, or A through F.

Multibyte Character Testing

Function	Header File	Page	Description
iswalnum()	wctype.h	150	Tests for wide alphanumeric characters.
iswalpha()	wctype.h	150	Tests for wide alphabetic characters.
iswcntrl()	wctype.h	150	Tests for wide control characters.
iswctype()	wctype.h	152	Tests for character property.
iswdigit()	wctype.h	150	Tests for wide decimal digits.
iswgraph()	wctype.h	150	Tests for wide printing characters excluding the space.

Function	Header File	Page	Description
iswlower()	wctype.h	150	Tests for wide lowercase letters.
iswprint()	wctype.h	150	Tests for wide printing characters.
iswpunct()	wctype.h	150	Tests for wide non-alphanumeric, non-space characters.
iswspace()	wctype.h	150	Tests for wide whitespace characters.
iswupper()	wctype.h	150	Tests for wide uppercase letters.
iswxdigit()	wctype.h	150	Tests for wide hexadecimal digits 0 through 9, a through f, or A through F.

Character Case Mapping

Function	Header File	Page	Description
tolower()	ctype.h	377	Converts a character to lowercase.
toupper()	ctype.h	377	Converts a character to uppercase.
towlower()	ctype.h	379	Converts a wide character to lowercase.
towupper()	ctype.h	379	Converts a wide character to uppercase.

Multibyte Character Manipulation

Function	Header File	Page	Description
btowc()	stdio.h wchar.h	54	Converts a single byte to a wide character.
mblen()	stdlib.h	172	Determines length of string of multibyte character.
mbrlen()	stdlib.h	173	Determines the length of a multibyte character. (Restartable version of mblen())
mbrtowc()	stdlib.h	176	Converts a multibyte character to a wide character. (Restartable version of mbtowc())
mbsinit()	stdlib.h	179	Tests state object for initial state.
mbsrtowcs()	stdlib.h	180	Converts a multibyte string to a wide character string. (Restartable version of mbstowcs())
mbstowcs()	stdlib.h	182	Converts a multibyte string to a wide <wchart.h>string.</wchart.h>
mbtowc()	stdlib.h	186	Converts multibyte characters to a wide <wchart.h>character.</wchart.h>
towctrans()	wctype.h	378	Translates wide character.
wcrtomb()	stdlib.h	396	Converts a wide character to a multibyte character. (Restartable version of wctomb()).

Function	Header File	Page	Description
wcsrtombs()	stdlib.h	418	Converts a wide character string to a multibyte character string. (Restartable version of wcstombs()).
wcstombs()	stdlib.h	426	Converts a wide character string to a multibyte character string.
wcscat()	wcstr.h	400	Concatenates wide character strings.
wcschr()	wcstr.h	402	Searches a wide character string for a wide character.
wcscmp()	wcstr.h	403	Compares two wide character strings.
wcscoll()	wcstr.h	404	Compares the locale-defined value of two wide-character strings.
wcscpy()	wcstr.h	405	Copies a wide character string.
wcscspn()	wcstr.h	406	Searches a wide character string for characters.
wcslen()	wcstr.h	409	Finds length of a wide character string.
wcsncat()	wcstr.h	410	Concatenates a wide character string segment.
wcsncmp()	wcstr.h	411	Compares wide character string segments.
wcsncpy()	wcstr.h	413	Copies wide character string segments.
wcspbrk()	wcstr.h	416	Locates wide characters in string.
wcsspn()	wcstr.h	420	Finds number of wide characters.
wcsrchr()	wcstr.h	417	Locates wide character in string.
wcswcs()	wcstr.h	432	Locates a wide character string in another wide character string.
wcswidth()	wchar.h	433	Determines the display width of a wide character string.
wcsxfrm()	wcstr.h	434	Transforms wide-character strings according to locale.
wctob()	stdlib.h	435	Converts a wide character to a single byte.
wctomb()	stdlib.h	436	Converts wide characters to multibyte characters.
wctrans()	wctype.h	437	Gets a handle for character mapping.
wctype()	wchar.h	438	Obtains a handle for character property classification.
wcwidth()	wchar.h	441	Determines the display width of a wide character.

Data Areas

Function	Header File	Page	Description
QXXCHGDA()	xxdtaa.h	218	Changes the data area.
QXXRTVDA()	xxdtaa.h		Retrieves a copy of the data area specified by dtname.

Message Catalogs

Function	Header File	Page	Description
catclose()	nl_types.h	57	Closes a message catalog.
catgets()	nl_types.h	58	Reads a message from an opened message catalog.
catopen()	nl_types.h	59	Opens a message catalog.

Regular Expression

Function	Header File	Page	Description
regcomp()	regex.h	237	Compiles regular expression.
regexec()	regex.h	240	Executes compiled regular expression.
regerror()	regex.h	238	Returns error message for regular expression.
regfree()	regex.h	242	Frees memory for regular expression.

abort() — Stop a Program

Format

#include <stdlib.h> void abort(void);

Language Level: ANSI

Thread Safe: YES.

Description

The abort() function causes an abnormal end of the program and returns control to the host environment. Like the exit() function, the abort() function deletes buffers and closes open files before ending the program.

Calls to the abort() function raise the SIGABRT signal. The abort() function will not result in the ending of the program if SIGABRT is caught by a signal handler, and the signal handler does not return.

Note: When compiled with SYSIFCOPT(*ASYNCSIGNAL), the abort() function cannot be called in a signal handler.

Return Value

There is no return value.

Example that uses abort()

This example tests for successful opening of the file myfile. If an error occurs, an error message is printed, and the program ends with a call to the abort() function.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
  FILE *stream;
  if ((stream = fopen("mylib/myfile", "r")) == NULL)
     perror("Could not open data file");
     abort();
```

Related Information

- "exit() End Program" on page 73
- "signal() Handle Interrupt Signals" on page 313
- "<stdlib.h>" on page 16
- See the signal() API in the System API Reference

abs() — Calculate Integer Absolute Value

Format

```
#include <stdlib.h>
int abs(int n);
```

Language Level: ANSI

Thread Safe: NO.

Description

The abs() function returns the absolute value of an integer argument n.

Return Value

There is no error return value. The result is undefined when the absolute value of the argument cannot be represented as an integer. The value of the minimum allowable integer is defined by INT_MIN in the limits.h> include file.

Example that uses abs()

This example calculates the absolute value of an integer x and assigns it to y.

```
#include <stdlib.h>
#include <stdio.h>
int main(void)
    int x = -4, y;
    y = abs(x);
```

- "fabs() Calculate Floating-Point Absolute Value" on page 74
- "labs() llabs() Calculate Absolute Value of Long and Long Long Integer" on page 155
- ""on page 7
- "<stdlib.h>" on page 16

acos() — Calculate Arccosine

Format

```
#include <math.h>
double acos(double x);
```

Language Level: ANSI

Thread Safe: NO.

Description

The acos() function calculates the arccosine of x, expressed in radians, in the range 0 to Π .

Return Value

The acos() function returns the arccosine of x. The value of x must be between -1 and 1 inclusive. If x is less than -1 or greater than 1, acos() sets errno to EDOM and returns 0.

Example that uses acos()

This example prompts for a value for x. It prints an error message if x is greater than 1 or less than -1; otherwise, it assigns the arccosine of x to y.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAX    1.0
#define MIN -1.0

int main(void)
{
    double x, y;
    printf( "Enter x\n" );
    scanf( "%lf", &x );

/* Output error if not in range */
    if ( x > MAX )
        printf( "Error: %lf too large for acos\n", x );
    else if ( x < MIN )
        printf( "Error: %lf too small for acos\n", x );
    else {</pre>
```

```
y = acos(x);
   printf( "acos( %lf ) = %lf\n", x, y );
/***** Expected output if 0.4 is entered: *******
Enter x
acos(0.400000) = 1.159279
```

- "asin() Calculate Arcsine" on page 43
- "atan() atan2() Calculate Arctangent" on page 45
- "cos() Calculate Cosine" on page 64
- "cosh() Calculate Hyperbolic Cosine" on page 65
- "sin() Calculate Sine" on page 315
- "sinh() Calculate Hyperbolic Sine" on page 316
- "tan() Calculate Tangent" on page 371
- "tanh() Calculate Hyperbolic Tangent" on page 372
- "<math.h>" on page 8

asctime() — Convert Time to Character String

Format

```
#include <time.h>
char *asctime(const struct tm *time);
```

Language Level: ANSI

Thread Safe: NO. Use asctime r() instead.

Description

The asctime() function converts time, stored as a structure pointed to by time, to a character string. You can obtain the time value from a call to the gmtime() function or the localtime() function; either returns a pointer to a tm structure defined in <time.h>.

The string result that asctime() produces contains exactly 26 characters and has the format:

```
"%.3s %.3s%3d %.2d:%.2d:%.2d %d\n"
```

The following are examples of the string returned:

```
Sat Jul 16 02:03:55 1994\n\0
Sat Jul 16 2:03:55 1994\n\0
```

The asctime() function uses a 24-hour-clock format. The days are abbreviated to: Sun, Mon, Tue, Wed, Thu, Fri, and Sat. The months are abbreviated to: Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, and Dec. All fields have constant width. Dates with only one digit are preceded either with a zero or a blank space. The new-line character (\n) and the null character (\0) occupy the last two positions of the string.

The time and date functions begin at 00:00:00 Universal Time, January 1, 1970.

Return Value

The asctime() function returns a pointer to the resulting character string. If the function is unsuccessful, it returns NULL.

Note: The asctime(), ctime() functions, and other time functions may use a common, statically allocated buffer to hold the return string. Each call to one of these functions may destroy the result of the previous call. The asctime_r(), ctime_r(), gmtime_r(), and localtime_r()functions do not use a common, statically-allocated buffer to hold the return string. These functions can be used in the place of the asctime(), ctime(), gmtime(), and localtime() functions if reentrancy is desired.

Example that uses asctime()

This example polls the system clock and prints a message that gives the current time.

Related Information

- "asctime_r() Convert Time to Character String (Restartable)" on page 42
- "ctime() Convert Time to Character String" on page 66
- "ctime_r() Convert Time to Character String (Restartable)" on page 67
- "gmtime() Convert Time" on page 141
- "gmtime_r() Convert Time (Restartable)" on page 143
- "localtime() Convert Time" on page 163
- "localtime_r() Convert Time (Restartable)" on page 164
- "mktime() Convert Local Time" on page 193
- "strftime() Convert Date/Time to String" on page 336
- "time() Determine Current Time" on page 373
- "printf() Print Formatted Characters" on page 200
- "setlocale() Set Locale" on page 308
- "<time.h>" on page 17

asctime_r() — Convert Time to Character String (Restartable)

Format

```
#include <time.h>
char *asctime r(const struct tm *tm, char *buf);
```

Thread Safe: NO.

Description

This function is the restartable version of the asctime() function.

The asctime_r() function converts time, stored as a structure pointed to by tm, to a character string. You can obtain the tm value from a call to gmtime r() or localtime_r(); either returns a pointer to a tm structure defined in <time.h>.

The string result that asctime_r() produces contains exactly 26 characters and has the format:

```
"%.3s %.3s%3d %.2d:%.2d:%.2d %d\n"
```

The following are examples of the string returned:

```
Sat Jul 16 02:03:55 1994\n\0
Sat Jul 16 2:03:55 1994\n\0
```

The asctime r() function uses a 24-hour-clock format. The days are abbreviated to: Sun, Mon, Tue, Wed, Thu, Fri, and Sat. The months are abbreviated to: Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, and Dec. All fields have constant width. Dates with only one digit are preceded either with a zero or a blank space. The new-line character (\n) and the null character (\0) occupy the last two positions of the string.

The time and date functions begin at 00:00:00 Universal Time, January 1, 1970.

Return Value

The asctime r() function returns a pointer to the resulting character string. If the function is unsuccessful, it returns NULL.

Example that uses asctime_r()

This example polls the system clock and prints a message giving the current time.

```
#include <time.h>
#include <stdio.h>
int main(void)
   struct tm *newtime;
    time t ltime;
    char mybuf[50];
/* Get the time in seconds */
    time(&ltime);
/* Convert it to the structure tm */
   newtime = localtime r(&ltime());
/* Print the local time as a string */
    printf("The current date and time are %s",
             asctime r(newtime, mybuf));
```

```
}
/******************
Output should be similar to *************
The current date and time are Fri Sep 16 132951 1994
*/
```

- "asctime() Convert Time to Character String" on page 40
- "ctime() Convert Time to Character String" on page 66
- "ctime_r() Convert Time to Character String (Restartable)" on page 67
- "gmtime() Convert Time" on page 141
- "gmtime_r() Convert Time (Restartable)" on page 143
- "localtime() Convert Time" on page 163
- "localtime_r() Convert Time (Restartable)" on page 164
- "mktime() Convert Local Time" on page 193
- "strftime() Convert Date/Time to String" on page 336
- "time() Determine Current Time" on page 373
- "printf() Print Formatted Characters" on page 200
- "<time.h>" on page 17

asin() — Calculate Arcsine

Format

```
#include <math.h>
double asin(double x);
```

Language Level: ANSI

Thread Safe: NO.

Description

The asin() function calculates the arcsine of x, in the range $-\pi/2$ to $\pi/2$ radians.

Return Value

The asin() function returns the arcsine of x. The value of x must be between -1 and 1. If x is less than -1 or greater than 1, the asin() function sets errno to EDOM, and returns a value of 0.

Example that uses asin()

This example prompts for a value for x. It prints an error message if x is greater than 1 or less than -1; otherwise, it assigns the arcsine of x to y.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAX 1.0
#define MIN -1.0

int main(void)
{
   double x, y;
```

```
printf( "Enter x\n" );
 scanf( "%1f", &x );
 /* Output error if not in range */
 if (x > MAX)
   printf( "Error: %lf too large for asin\n", x );
 else if (x < MIN)
   printf( "Error: %lf too small for asin\n", x );
   y = asin(x);
   printf( "asin( %lf ) = %lf\n", x, y );
/****** Output should be similar to *********
asin(0.200000) = 0.201358
```

- "acos() Calculate Arccosine" on page 39
- "atan() atan2() Calculate Arctangent" on page 45
- "cos() Calculate Cosine" on page 64
- "cosh() Calculate Hyperbolic Cosine" on page 65
- "sin() Calculate Sine" on page 315
- "sinh() Calculate Hyperbolic Sine" on page 316
- "tan() Calculate Tangent" on page 371
- "tanh() Calculate Hyperbolic Tangent" on page 372
- "<math.h>" on page 8

assert() — Verify Condition

Format

#include <assert.h> void assert(int expression);

Language Level: ANSI

Thread Safe: NO.

Description

The assert() function prints a diagnostic message to stderr and aborts the program if *expression* is false (zero). The diagnostic message has the format: Assertion failed: expression, file filename, line line-number.

The assert() function takes no action if the *expression* is true (nonzero).

Use the assert() function to identify program logic errors. Choose an expression that holds true only if the program is operating as you intend. After you have debugged the program, you can use the special no-debug identifier NDEBUG to remove the assert() calls from the program. If you define NDEBUG to any value with a #define directive, the C preprocessor expands all assert calls to void expressions. If you use NDEBUG, you must define it before you include <assert.h> in the program.

Return Value

There is no return value.

Note: The assert() function is defined as a macro. Do not use the #undef directive with assert().

Example that uses assert()

In this example, the assert() function tests *string* for a null string and an empty string, and verifies that *length* is positive before processing these arguments.

```
#include <stdio.h>
#include <assert.h>
void analyze (char *, int);
int main(void)
  char *string = "ABC";
  int length = 3;
  analyze(string, length);
  printf("The string %s is not null or empty, "
         "and has length %d n", string, length);
void analyze(char *string, int length)
  assert(string != NULL);
                             /* cannot be NULL */
  assert(*string != '\0');
                           /* cannot be empty */
  assert(length > 0);
                            /* must be positive */
/****** Output should be similar to *********
The string ABC is not null or empty, and has length 3
```

Related Information

- "abort() Stop a Program" on page 37
- "<assert.h>" on page 3
- See "#define" under Preprocessor Directives in the ILE C for AS/400 Language Reference.

atan() - atan2() - Calculate Arctangent

Format

```
#include <math.h>
double atan(double x);
double atan2(double y, double x);
```

Language Level: ANSI

Description

The atan() and atan2() functions calculate the arctangent of x and y/x, respectively.

Return Value

The atan() function returns a value in the range $-\pi/2$ to $\pi/2$ radians. The atan2() function returns a value in the range $-\pi$ to π radians. If both arguments of the atan2() function are zero, the function sets errno to EDOM, and returns a value of

Example that uses atan()

```
This example calculates arctangents using the atan() and atan2() functions.
```

```
#include <math.h>
#include <stdio.h>
int main(void)
   double a,b,c,d;
   c = 0.45;
   d = 0.23;
   a = atan(c);
   b = atan2(c,d);
   printf("atan(%lf) = %lf/n", c, a);
   printf("atan2( %lf, %lf ) = %lf/n", c, d, b);
/****** Output should be similar to **********
atan(0.450000) = 0.422854
atan2( 0.450000, 0.230000 ) = 1.098299
```

Related Information

- "acos() Calculate Arccosine" on page 39
- "asin() Calculate Arcsine" on page 43
- "cos() Calculate Cosine" on page 64
- "cosh() Calculate Hyperbolic Cosine" on page 65
- "sin() Calculate Sine" on page 315
- "sinh() Calculate Hyperbolic Sine" on page 316
- "tan() Calculate Tangent" on page 371
- "tanh() Calculate Hyperbolic Tangent" on page 372
- "<math.h>" on page 8

atexit() — Record Program Ending Function

Format

```
#include <stdlib.h>
int atexit(void (*func)(void));
```

Language Level: ANSI

Description

The atexit() function records the function, pointed to by *func*, that the system calls at normal program end. For portability, you should use the atexit() function to register a maximum of 32 functions. The functions are processed in a last-in, first-out order. The atexit() function cannot be called from the OPM default

activation group. Most functions can be used with the atexit function; however, if the exit function is used the atexit function will fail.

Return Value

The atexit() function returns 0 if it is successful, and nonzero if it fails.

Example that uses atexit()

```
This example uses the atexit() function to call goodbye() at program end.
```

Related Information

- "exit() End Program" on page 73
- "signal() Handle Interrupt Signals" on page 313
- "<stdlib.h>" on page 16

atof() — Convert Character String to Float

Format

```
#include <stdlib.h>
double atof(const char *string);
```

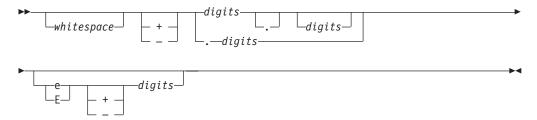
Language Level: ANSI

Description

The atof() function converts a character string to a double-precision floating-point value.

The input *string* is a sequence of characters that can be interpreted as a numeric value of the specified return type. The function stops reading the input string at the first character that it cannot recognize as part of a number. This character can be the null character that ends the string.

The atof() function expects a *string* in the following form:



The white space consists of the same characters for which the isspace() function is true, such as spaces and tabs. The atof() function ignores leading white-space characters.

For the atof() function, digits is one or more decimal digits; if no digits appear before the decimal point, at least one digit must appear after the decimal point. The decimal digits can precede an exponent, introduced by the letter e or E. The exponent is a decimal integer, which may be signed.

The atof() function will not fail if a character other than a digit follows an E or if e is read in as an exponent. For example, 100elf will be converted to the floating-point value 100.0. The accuracy is up to 17 significant character digits.

Return Value

The atof() function returns a double value that is produced by interpreting the input characters as a number. The return value is 0 if the function cannot convert the input to a value of that type. The return value is undefined in case of overflow.

Example that uses atof()

This example shows how to convert numbers that are stored as strings to numeric values.

```
#include <stdlib.h>
#include <stdio.h>
int main(void)
   double x;
   char *s;
   s = "-2309.12E-15";
                /* x = -2309.12E-15 */
   x = atof(s);
   printf("x = %.4e\n",x);
/******* Output should be similar to: *********
x = -2.3091e-12
```

Related Information

- "atoi() Convert Character String to Integer" on page 49
- "atol() atoll() Convert Character String to Long or Long Long Integer" on page 50

- "strtol() strtoll() Convert Character String to Long and Long Long Integer" on page 362
- "strtod() Convert Character String to Double" on page 357
- "<stdlib.h>" on page 16

atoi() — Convert Character String to Integer

Format

```
#include <stdlib.h>
int atoi(const char *string);
```

Language Level: ANSI

Description

The atoi() function converts a character string to an integer value. The input *string* is a sequence of characters that can be interpreted as a numeric value of the specified return type. The function stops reading the input string at the first character that it cannot recognize as part of a number. This character can be the null character that ends the string.

The atoi() function does not recognize decimal points nor exponents. The string argument for this function has the form:



where *whitespace* consists of the same characters for which the isspace() function is true, such as spaces and tabs. The atoi()function ignores leading white-space characters. The value *digits* represents one or more decimal digits.

Return Value

The atoi() function returns an int value that is produced by interpreting the input characters as a number. The return value is 0 if the function cannot convert the input to a value of that type. The return value is undefined in the case of an overflow.

Example that uses atoi()

This example shows how to convert numbers that are stored as strings to numeric values.

```
/******** Output should be similar to: ********
i = -9885
```

- "atof() Convert Character String to Float" on page 47
- "atol() atoll() Convert Character String to Long or Long Long Integer"
- "strtod() Convert Character String to Double" on page 357
- "strtol() strtoll() Convert Character String to Long and Long Long Integer" on page 362
- "<stdlib.h>" on page 16

atol() — atoll() — Convert Character String to Long or Long Long Integer

```
Format (atol())
#include <stdlib.h>
long int atol(const char *string);
Format (atoll())
#include <stdlib.h>
long long int atoll(const char *string);
```

Description

Language Level: ANSI

The atol() function converts a character string to a long value. The atoll() function converts a character string to a long long value.

The input *string* is a sequence of characters that can be interpreted as a numeric value of the specified return type. The function stops reading the input string at the first character that it cannot recognize as part of a number. This character can be the null character that ends the string.

The atol() and atoll() functions do not recognize decimal points nor exponents. The *string* argument for this function has the form:



where whitespace consists of the same characters for which the isspace() function is true, such as spaces and tabs. The atol() and atoll() functions ignore leading white-space characters. The value digits represents one or more decimal digits.

Return Value

The atol() and atoll() functions return a long or a long long value that is produced by interpreting the input characters as a number. The return value is 0L if the function cannot convert the input to a value of that type. The return value is undefined in case of overflow.

Example that uses atol()

This example shows how to convert numbers that are stored as strings to numeric values.

```
#include <stdlib.h>
#include <stdio.h>
int main(void)
   long 1;
   char *s;
   s = "98854 dollars";
   1 = atol(s); /*1 = 98854 */
   printf("1 = %.ld\n",1);
/************* Output should be similar to: *********
1 = 98854
*/
```

Related Information

- "atof() Convert Character String to Float" on page 47
- "atoi() Convert Character String to Integer" on page 49
- "strtod() Convert Character String to Double" on page 357
- "strtol() strtoll() Convert Character String to Long and Long Long Integer" on page 362
- "<stdlib.h>" on page 16

Bessel Functions

Format

```
#include <math.h>
double j0(double x);
double j1(double x);
double jn(int n, double x);
double y0(double x);
double y1(double x);
double yn(int n, double x);
```

Language Level: ILE C Extension

Description

Bessel functions solve certain types of differential equations. The j0(), j1(), and jn() functions are Bessel functions of the first kind for orders 0, 1, and n, respectively. The y0(), y1(), and yn() functions are Bessel functions of the second kind for orders 0, 1, and n, respectively.

The argument x must be positive. The argument n should be greater than or equal to zero. If n is less than zero, it will be a negative exponent.

Return Value

For j0(), j1(), y0(), or y1(), if the absolute value of x is too large, the function sets errno to ERANGE, and returns 0. For y0(), y1(), or yn(), if x is negative, the function sets errno to EDOM and returns the value -HUGE VAL. For y0, y1(), or yn(), if x causes overflow, the function sets errno to ERANGE and returns the value -HUGE_VAL.

Example that uses Bessel Functions

This example computes y to be the order 0 Bessel function of the first kind for x. It also computes z to be the order 3 Bessel function of the second kind for x.

```
#include <math.h>
#include <stdio.h>
int main(void)
   double x, y, z;
   x = 4.27;
   y = j0(x);
                   /* y = -0.3660 is the order 0 bessel */
                   /* function of the first kind for x */
   z = yn(3,x);
                  /* z = -0.0875 is the order 3 bessel */
                   /* function of the second kind for x */
   printf("y = %1f\n", y);
   printf("z = %lf\n", z);
/****** Output should be similar to: **********
y = -0.366022
z = -0.087482
```

Related Information

- "erf() erfc() Calculate Error Functions" on page 72
- "gamma() Gamma Function" on page 131
- "<math.h>" on page 8

bsearch() — Search Arrays

Format

```
#include <stdlib.h>
void *bsearch(const void *key, const void *base,
              size t num, size t size,
              int (*compare)(const void *key, const void *element));
```

Language Level: ANSI

Description

The bsearch() function performs a binary search of an array of *num* elements, each of size bytes. The array must be sorted in ascending order by the function pointed to by *compare*. The *base* is a pointer to the base of the array to search, and *key* is the value being sought.

The *compare* argument is a pointer to a function you must supply that compares two items and returns a value specifying their relationship. The first item in the argument list of the compare() function is the pointer to the value of the item that is being searched for. The second item in the argument list of the compare() function is a pointer to the array element being compared with the key. The compare() function must compare the key value with the array element and then return one of the following values:

Value	Meaning
Less than 0	key less than element
0	key identical to element
Greater than 0	key greater than element

Return Value

The bsearch() function returns a pointer to *key* in the array to which *base* points. If two keys are equal, the element that *key* will point to is unspecified. If the bsearch() function cannot find the *key*, it returns NULL.

Example that uses bsearch()

This example performs a binary search on the argv array of pointers to the program parameters and finds the position of the argument PATH. It first removes the program name from argv, and then sorts the array alphabetically before calling bsearch(). The compare1() and compare2() functions compare the values pointed to by arg1 and arg2 and return the result to the bsearch() function.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int compare1(const void *, const void *);
int compare2(const void *, const void *);
main(int argc, char *argv∏)
                        /* This program performs a binary
  char **result;
                        /* search on the argv array of pointers */
  char *key = "PATH";
                        /* to the program parameters. It first */
  int i;
                         /* removes the program name from argv
                                                                  */
                         /* then sorts the array alphabetically
                                                                  */
  argv++;
                         /* before calling bsearch.
  argc--;
  qsort((char *)argv, argc, sizeof(char *), compare1);
  result = (char**)bsearch(&key, (char *)argv, argc, sizeof(char *), compare2);
  if (result != NULL) {
      printf("result =<%s>\n",*result);
  else printf("result is null\n");
         /*This function compares the values pointed to by arg1 \*/
         /*and arg2 and returns the result to qsort. arg1 and */
         /*arg2 are both pointers to elements of the argv array. */
int compare1(const void *arg1, const void *arg2)
    return (strcmp(*(char **)arg1, *(char **)arg2));
         /*This function compares the values pointed to by arg1 */
         /*and arg2 and returns the result to bsearch
         /*arg1 is a pointer to the key value, arg2 points to
         /*the element of argv that is being compared to the key */
         /*value.
int compare2(const void *arg1, const void *arg2)
    return (strcmp(*(char **)arg1, *(char **)arg2));
```

```
/********* Output should be similar to: *******
result = <PATH>
********* When the input on the iSeries command line is ******
CALL BSEARCH PARM(WHERE IS PATH IN THIS PHRASE'?')
```

- "qsort() Sort Array" on page 216
- "<stdlib.h>" on page 16

btowc() — Convert Single Byte to Wide Character

Format

```
#include <stdio.h>
#include <wchar.h>
wint t btowc(int c);
```

Language Level: ANSI

Description

The btowc() function converts the single byte value c to the wide-character representation of c. If c does not constitute a valid (one-byte) multibyte character in the initial shift state, the btowc() function returns WEOF.

The behavior of the btowc() function is affected by the LC CTYPE category of the current locale. This function is available only when LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

Note: If a program is compiled with LOCALETYPE(*LOCALEUCS2), the btowc() function will behave as if the program was compiled with LOCALETYPE(*LOCALE), because the btowc() function does not support UNICODE.

Return Value

The btowc() function returns WEOF if c has the value EOF, or if (unsigned char) c does not constitute a valid (one-byte) multibyte character in the initial shift state. Otherwise, it returns the wide-character representation of that character.

Example that uses btowc()

This example scans various types of data.

```
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <local.h>
#define UPPER LIMIT
                      0xFF
int main(void)
   int wc;
   int ch;
```

```
if (NULL == setlocale(LC ALL, "")) {
   printf("Locale could not be loaded\n");
   exit(1);
 for (ch = 0; ch <= UPPER LIMIT; ++ch) {</pre>
   wc = btowc(ch);
   if (wc==WEOF)
      printf("%#04x is not a one-byte multibyte character\n", ch);
      printf("%#04x has wide character representation: %#06x\n", ch, wc);
wc = btowc(EOF);
 if (wc==WEOF) {
   printf("The character is EOF.\n", ch);
   printf("EOF has wide character representation: %#06x\n", wc);
return 0;
/****************************
   If the locale is bound to SBCS, the output should be similar to:
   0000 has wide character representation: 000000
   0x01 has wide character representation: 0x0001
   Oxfe has wide character representation: 0x00fe
   Oxff has wide character representation: 0x00ff
   The character is EOF.
   If the locale is bound to DBCS, the output should be similar to:
   0000 has wide character representation: 000000
   0x01 has wide character representation: 0x0001
   0x80 has wide character representation: 0x80
   0x83 is not a one-byte multibyte character
   0xa2 has wide character representation: 0xa2
   Oxff has wide character representation: 0x00ff
   The character is EOF.
```

- "mblen() Determine Length of a Multibyte Character" on page 172
- "mbtowc() Convert Multibyte Character to a Wide Character" on page 186
- "mbrtowc() Convert a Multibyte Character to a Wide Character (Restartable)" on page 176
- "mbsrtowcs() Convert a Multibyte String to a Wide Character String (Restartable)" on page 180
- "setlocale() Set Locale" on page 308
- "wcrtomb() Convert a Wide Character to a Multibyte Character (Restartable)" on page 396
- "wcsrtombs() Convert Wide Character String to Multibyte String (Restartable)" on page 418
- "<stdio.h>" on page 15
- "<wchar.h>" on page 18

_C_Get_Ssn_Handle() — Handle to C Session

Format

```
#include <stdio.h>
```

```
SSN HANDLE T C Get Ssn Handle (void)
```

Language Level: ILE C Extension

Description

Returns a handle to the C session for use with Dynamic Screen Manager (DSM) APIs.

Return Value

The _C_Get_Ssn_Handle() function returns a handle to the C session. If an error occurs, _SSN_HANDLE_T is set to zero. See the API topic for more information about using the _C_Get_Ssn_Handle() function with DSM APIs.

calloc() — Reserve and Initialize Storage

Format

```
#include <stdlib.h>
void *calloc(size_t num, size_t size);
```

Language Level: ANSI

Thread Safe: YES

Description

The calloc() function reserves storage space for an array of *num* elements, each of length *size* bytes. The calloc() function then gives all the bits of each element an initial value of 0.

Return Value

The calloc() function returns a pointer to the reserved space. The storage space to which the return value points is suitably aligned for storage of any type of object. To get a pointer to a type, use a type cast on the return value. The return value is NULL if there is not enough storage, or if *num* or *size* is 0.

Note: To use **Teraspace** storage instead of heap storage without changing the C source code, specify the TERASPACE(*YES *TSIFC) parameter on the compiler command. This maps the calloc() library function to C TS calloc(), its Teraspace storage counterpart. The maximum amount of Teraspace storage that can be allocated by each call to _C_TS_calloc() is 2GB - 224, or 2147483424 bytes.

For more information about Teraspace, see the ILE Concepts manual.

Example that uses calloc()

This example prompts for the number of array entries required, and then reserves enough space in storage for the entries. If calloc() is successful, the example prints out each entry; otherwise, it prints out an error.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
 long * array;
                                          /* start of the array
  */
                                           /* index variable
 long * index;
  */
 int
                                           /* index variable
        i;
  */
 int num;
                                          /* number of entries of the array
  printf( "Enter the size of the array\n" );
  scanf( "%i", &num);
                                            /* allocate num entries */
  if ( (index = array = (long *) calloc( num, sizeof( long ))) != NULL )
    for ( i = 0; i < num; ++i )
                                           /* put values in arr
       *index++ = i;
                                           /* using pointer no
                                                                    */
   for ( i = 0; i < num; ++i )
                                           /* print the array out */
      printf( "array[%i ] = %i\n", i, array[i] );
  else
  { /* out of storage */
   perror( "Out of storage" );
   abort();
 }
/************ Output should be similar to: ***********
Enter the size of the array
array[ 0 ] = 0
array[ 1 ] = 1
array[2] = 2
```

- "free() Release Storage Blocks" on page 109
- "malloc() Reserve Storage Block" on page 170
- "realloc() Change Reserved Storage Block Size" on page 234
- "<stdlib.h>" on page 16

catclose() — Close Message Catalog

Format

#include <nl_types.h>
int catclose (nl_catd catd);

Language Level: XPG4

Thread Safe: YES.

Description

The catclose() function closes the previously opened message catalog that is identified by *catd*.

Note: This function is only accessible when LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) and SYSIFCOPT(*IFSIO) are specified on the compilation command.

Return Value

If the close is performed successfully, 0 is returned. Otherwise, -1 is returned indicating failure, which may happen if catd is not a valid message catalog descriptor.

The value of errno may be set to:

EBADF

The catalog descriptor is not valid.

EINTR

The function was interrupted by a signal.

Example that uses catclose()

```
#include <stdio.h>
#include <nl types.h>
#include <locale.h>
/* Name of the message catalog is "/qsys.lib/mylib.lib/msgs.usrspc" */
int main(void) {
  nl catd msg file;
  char * my msg;
  char * my_locale;
  setlocale(LC ALL, NULL);
  msg_file = catopen("/qsys.lib/mylib.lib/msgs.usrspc", 0);
  if (msg_file != CATD_ERR) {
    my_msg = catgets(msg_file, 1, 2, "oops");
    printf("%s\n", my_msg);
     catclose(msg_file);
   }
```

Related Information

- "catopen() Open Message Catalog" on page 59
- "catgets() Retrieve a Message from a Message Catalog"

catgets() — Retrieve a Message from a Message Catalog

Format

```
#include <nl types.h>
char *catgets(nl catd catd, int set id, int msg id, char *s);
```

Language Level: XPG4

Thread Safe: YES.

Description

The catgets() function retrieves message *msg_id*, in set *set_id* from the message catalog that is identified by *catd*. *catd* is a message catalog descriptor that is returned by a previous call to catopen(). The *s* argument points to a default message which will be returned by catgets() if the identified message cannot be retrieved.

Note: This function is only accessible when LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) and SYSIFCOPT(*IFSIO) are specified on the compilation command.

Return Value

If the message is retrieved successfully, then catgets () returns a pointer to the message string that is contained in the message catalog. If the message is retrieved unsuccessfully, then a pointer to the default string s is returned.

The value of errno may be set to the following:

EBADF

The catalog descriptor is not valid.

EINTR

The function was interrupted by a signal.

Example that uses catgets()

```
#include <stdio.h>
#include <nl_types.h>
#include <locale.h>

/* Name of the message catalog is "/qsys.lib/mylib.lib/msgs.usrspc" */
int main(void) {
    nl_catd msg_file;
    char * my_msg;
    char * my_locale;

    setlocale(LC_ALL, NULL);
    msg_file = catopen("/qsys.lib/mylib.lib/msgs.usrspc", 0);

    if (msg_file != CATD_ERR) {
        my_msg = catgets(msg_file, 1, 2, "oops");
        printf("%s\n", my_msg);
        catclose(msg_file);
    }
}
```

Related Information

- "catclose() Close Message Catalog" on page 57
- "catopen() Open Message Catalog"

catopen() — Open Message Catalog

Format

```
#include <nl types.h>
nl catd catopen(const char *name, int oflag);
```

Language Level: XPG4

Thread Safe: YES.

Description

The catopen() function opens a message catalog, which must be done before a message can be retrieved. The NLSPATH environment variable and the LC_MESSAGES category are used to find the specified message catalog if no slash (/) characters are found in the name. If the name contains one or more slash (/) characters, then the name is interpreted as a path name of the catalog to open.

If there is no NLSPATH environment variable, or if a message catalog cannot be found in the path specified by NLSPATH, then a default path will be used. The default path may be affected by the setting of the LC_MESSAGES locale category if the value of oflag is NL CAT LOCALE, or by the LANG environment variable if the value of *oflag* is zero.

The message catalog descriptor will remain valid until it is closed by a call to catclose(). If the LC_MESSAGES locale category is changed, it may invalidate existing open message catalogs.

Note: This function is only accessible when LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) and SYSIFCOPT(*IFSIO) are specified on the compilation command. The name of the message catalog must be a valid Integrated File System file name.

The catopen() function might fail under the following conditions, and the value of errno may be set to:

EMFILE

NL_MAXOPEN message catalogs are currently open.

EACCES

Insufficient authority to read the message catalog specified, or to search the component of the path prefix of the message catalog specified.

ENAMETOOLONG

The length of the path name of the message catalog exceeds PATH_MAX, or a path name component is longer than NAME_MAX.

ENFILE

Too many files are currently open in the system.

ENOENT

The message catalog does not exist, or the name argument points to an empty string.

Return Value

If the message catalog is opened successfully, then a valid catalog descriptor is returned. If catopen() is unsuccessful, then it returns CATD ERR ((nl catd)-1).

Example that uses catopen()

```
#include <stdio.h>
#include <nl_types.h>
#include <locale.h>

/* Name of the message catalog is "/qsys.lib/mylib.lib/msgs.usrspc" */
int main(void) {
    nl_catd msg_file;
    char * my_msg;
    char * my_locale;

    setlocale(LC_ALL, NULL);
    msg_file = catopen("/qsys.lib/mylib.lib/msgs.usrspc", 0);

    if (msg_file != CATD_ERR) {
        my_msg = catgets(msg_file, 1, 2, "oops");
        printf("%s\n", my_msg);
        catclose(msg_file);
    }
}
```

Related Information

- "catclose() Close Message Catalog" on page 57
- "catgets() Retrieve a Message from a Message Catalog" on page 58

ceil() — Find Integer >=Argument

Format

```
#include <math.h>
double ceil(double x);
```

Language Level: ANSI

Description

The ceil() function computes the smallest integer that is greater than or equal to x.

Return Value

The ceil() function returns the integer as a double value.

Example that uses ceil()

This example sets y to the smallest integer greater than 1.05, and then to the smallest integer greater than -1.05. The results are 2.0 and -1.0, respectively.

```
#include <math.h>
#include <stdio.h>
int main(void)
  double y, z;
                     /* y = 2.0 */
/* z = -1.0 */
  y = ceil(1.05);
  z = ceil(-1.05);
  printf("y = %.2f; z = %.2f\n", y, z);
/************* Output should be similar to: ***************
y = 2.00 ; z = -1.00
```

Related Information

- "floor() —Find Integer <= Argument" on page 91
- "fmod() Calculate Floating-Point Remainder" on page 91
- "<math.h>" on page 8

clearerr() — Reset Error Indicators

Format

```
#include <stdio.h>
void clearerr (FILE *stream);
```

Language Level: ANSI

Description

The clearerr() function resets the error indicator and end-of-file indicator for the specified stream. Once set, the indicators for a specified stream remain set until your program calls the clearerr() function or the rewind() function. The fseek() function also clears the end-of-file indicator. The ILE C/C++ run-time environment does not automatically clear error or end of file indicators.

Return Value

There is no return value.

The value of errno may be set to:

Value Meaning

EBADF

The file pointer or descriptor is not valid.

ENOTOPEN

The file is not open.

ESTDIN

stdin cannot be opened.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

Example that uses clearerr()

This example reads a data stream, and then checks that a read error has not occurred.

```
#include <stdio.h>
#include <stdlib.h>

FILE *stream;
int c;

int main(void)
{
   if ((stream = fopen("mylib/myfile", "r")) != NULL)
   {
      if ((c=getc(stream)) == EOF)
      {
        if (ferror(stream))
        {
            perror("Read error");
            clearerr(stream);
        }
    }
   }
   else
    exit(0);
}
```

Related Information

- "feof() Test End-of-File Indicator" on page 79
- "ferror() Test for Read/Write Errors" on page 79
- "fseek() fseeko() Reposition File Position" on page 114
- "perror() Print Error Message" on page 198
- "rewind() Adjust Current File Position" on page 245
- "strerror() Set Pointer to Run-Time Error Message" on page 333
- "<stdio.h>" on page 15

clock() — Determine Processor Time

Format

```
#include <time.h>
clock_t clock(void);
```

Language Level: ANSI

Description

The clock() function returns an approximation of the processor time used by the program since the beginning of an implementation-defined time-period that is related to the process invocation. To obtain the time in seconds, divide the value that is returned by clock() by the value of the macro CLOCKS_PER_SEC.

Return Value

If the value of the processor time is not available or cannot be represented, the clock() function returns the value (clock_t)-1.

To measure the time spent in a program, call clock() at the start of the program, and subtract its return value from the value returned by subsequent calls to clock(). On other platforms, you can not always rely on the clock()function because calls to the system() function may reset the clock.

Example that uses clock()

This example prints the time that has elapsed since the program was called.

```
#include <time.h>
#include <stdio.h>
double time1, timedif;
                        /* use doubles to show small values */
int main(void)
   int i;
   /* running the FOR loop 10000 times */
   for (i=0; i<10000; i++);
   /* call clock a second time */
   timedif = ( ((double) clock()) / CLOCKS_PER SEC) - time1;
   printf("The elapsed time is %lf seconds\n", timedif);
```

Related Information

- "difftime() Compute Time Difference" on page 69
- "time() Determine Current Time" on page 373
- "<time.h>" on page 17

cos() — Calculate Cosine

Format

```
#include <math.h>
double cos(double x);
```

Language Level: ANSI

Description

The cos() function calculates the cosine of x. The value x is expressed in radians. If x is too large, a partial loss of significance in the result may occur.

Return Value

The cos() function returns the cosine of x. The value of errno may be set to either EDOM or ERANGE.

Example that uses cos()

This example calculates y to be the cosine of x.

Related Information

- "acos() Calculate Arccosine" on page 39
- "cosh() Calculate Hyperbolic Cosine"
- "sin() Calculate Sine" on page 315
- "sinh() Calculate Hyperbolic Sine" on page 316
- "tan() Calculate Tangent" on page 371
- "tanh() Calculate Hyperbolic Tangent" on page 372
- "<math.h>" on page 8

cosh() — Calculate Hyperbolic Cosine

Format

#include <math.h>
double cosh(double x);

Language Level: ANSI

Description

The cosh() function calculates the hyperbolic cosine of x. The value x is expressed in radians.

Return Value

The cosh() function returns the hyperbolic cosine of x. If the result is too large, cosh() returns the value HUGE_VAL and sets errno to ERANGE.

Example that uses cosh()

This example calculates y to be the hyperbolic cosine of x.

```
#include <math.h>
#include <stdio.h>
int main(void)
  double x,y;
  x = 7.2;
  y = cosh(x);
  printf("cosh(%lf) = %lf\n", x, y);
/******* Output should be similar to: *********
cosh(7.200000) = 669.715755
```

Related Information

- "acos() Calculate Arccosine" on page 39
- "cos() Calculate Cosine" on page 64
- "sin() Calculate Sine" on page 315
- "sinh() Calculate Hyperbolic Sine" on page 316
- "tan() Calculate Tangent" on page 371
- "tanh() Calculate Hyperbolic Tangent" on page 372
- "<math.h>" on page 8

ctime() — Convert Time to Character String

Format

```
#include <time.h>
char *ctime(const time_t *time);
```

Language Level: ANSI

Thread Safe: NO. Use ctime_r() instead.

Description

The ctime() function converts the time value pointed to by time to local time in the form of a character string. A time value is usually obtained by a call to the time() function.

The string result that is produced by ctime() contains exactly 26 characters and has the format:

```
"%.3s %.3s%3d %.2d:%.2d:%.2d %d\n"
```

For example:

```
Mon Jul 16 02:03:55 1987\n\0
```

The ctime() function uses a 24-hour clock format. The days are abbreviated to: Sun, Mon, Tue, Wed, Thu, Fri, and Sat. The months are abbreviated to: Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, and Dec. All fields have a constant width. Dates with only one digit are preceded with a zero. The new-line character (\n) and the null character (\0) occupy the last two positions of the string.

Return Value

The ctime() function returns a pointer to the character string result. If the function is unsuccessful, it returns NULL. A call to the ctime() function is equivalent to: asctime(localtime(&anytime))

Note: The asctime() and ctime() functions, and other time functions may use a common, statically allocated buffer to hold the return string. Each call to one of these functions may destroy the result of the previous call. The asctime r(), ctime r(), gmtime r(), and localtime r() functions do not use a common, statically-allocated buffer to hold the return string. These functions can be used in the place of asctime(), ctime(), gmtime(), and localtime() if reentrancy is desired.

Example that uses ctime()

This example polls the system clock using time(). It then prints a message giving the current date and time.

```
#include <time.h>
#include <stdio.h>
int main(void)
   time t ltime;
   time(localtime());
  printf("the time is %s", ctime(localtime()));
```

Related Information

- "asctime() Convert Time to Character String" on page 40
- "asctime_r() Convert Time to Character String (Restartable)" on page 42
- "ctime_r() Convert Time to Character String (Restartable)"
- "gmtime() Convert Time" on page 141
- "gmtime_r() Convert Time (Restartable)" on page 143
- "localtime() Convert Time" on page 163
- "localtime r() Convert Time (Restartable)" on page 164
- "mktime() Convert Local Time" on page 193
- "setlocale() Set Locale" on page 308
- "strftime() Convert Date/Time to String" on page 336
- "time() Determine Current Time" on page 373
- "printf() Print Formatted Characters" on page 200
- "<time.h>" on page 17

ctime_r() — Convert Time to Character String (Restartable)

Format

```
#include <time.h>
char *ctime_r(const time_t *time, char *buf);
```

Description

This function is the restartable version of the ctime() function.

The ctime r() function converts the time value pointed to by time to local time in the form of a character string. A time value is usually obtained by a call to the time() function.

The string result that is produced by the ctime r() function contains exactly 26 characters and has the format:

```
"%.3s %.3s%3d %.2d:%.2d:%.2d %d\n"
```

For example:

```
Mon Jul 16 02:03:55 1987\n\0
```

The ctime r() function uses a 24-hour clock format. The days are abbreviated to: Sun, Mon, Tue, Wed, Thu, Fri, and Sat. The months are abbreviated to: Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, and Dec. All fields have a constant width. Dates with only one digit are preceded with a zero. The new-line character (\n) and the null character (\0) occupy the last two positions of the string.

Return Value

The ctime r() function returns a pointer to the character string result. If the function is unsuccessful, it returns NULL. A call to ctime_r() is equivalent to: asctime r(localtime r(&anytime, buf), buf)

where buf is a pointer to char.

Example that uses ctime r()

This example polls the system clock using ctime r(). It then prints a message giving the current date and time.

```
#include <time.h>
#include <stdio.h>
int main(void)
  time t ltime;
  char buf[50];
   time(localtime());
   printf("the time is %s", ctime r(localtime(), buf));
```

Related Information

- "asctime() Convert Time to Character String" on page 40
- "asctime_r() Convert Time to Character String (Restartable)" on page 42
- "ctime() Convert Time to Character String" on page 66
- "gmtime() Convert Time" on page 141
- "gmtime_r() Convert Time (Restartable)" on page 143
- "localtime_r() Convert Time (Restartable)" on page 164
- "localtime() Convert Time" on page 163
- "mktime() Convert Local Time" on page 193
- "strftime() Convert Date/Time to String" on page 336
- "time() Determine Current Time" on page 373
- "<time.h>" on page 17

difftime() — Compute Time Difference

Format

#include <time.h>
double difftime(time_t time2, time_t time1);

Language Level: ANSI

Description

The difftime() function computes the difference in seconds between *time*2 and *time*1.

Return Value

The difftime() function returns the elapsed time in seconds from *time1* to *time2* as a double precision number. Type time_t is defined in <time.h>.

Example that uses difftime()

This example shows a timing application that uses difftime(). The example calculates how long, on average, it takes to find the prime numbers from 2 to 10000.

```
#include <time.h>
#include <stdio.h>
#define RUNS 1000
#define SIZE 10000
int mark[SIZE];
int main(void)
  time t start, finish;
  int \overline{i}, loop, n, num;
  time(&start);
   /st This loop finds the prime numbers between 2 and SIZE st/
  for (loop = 0; loop < RUNS; ++loop)</pre>
     for (n = 0; n < SIZE; ++n)
       mark[n] = 0;
     /* This loops marks all the composite numbers with -1 */
     for (num = 0, n = 2; n < SIZE; ++n)
        if (! mark[n])
           for (i = 2 * n; i < SIZE; i += n)
               mark[i] = -1;
           ++num;
  time(&finish);
  printf("Program takes an average of %f seconds "
                 "to find %d primes.\n",
                  difftime(finish,start)/RUNS, num);
/******* Output should be similar: *********
The program takes an average of 0.106000 seconds to find 1229 primes.
*/
Related Information
• "asctime() — Convert Time to Character String" on page 40
• "asctime_r() — Convert Time to Character String (Restartable)" on page 42
• "ctime() — Convert Time to Character String" on page 66
• "ctime_r() — Convert Time to Character String (Restartable)" on page 67
• "gmtime() — Convert Time" on page 141
• "gmtime_r() — Convert Time (Restartable)" on page 143
• "localtime() — Convert Time" on page 163
• "localtime_r() — Convert Time (Restartable)" on page 164
• "mktime() — Convert Local Time" on page 193

    "strftime() — Convert Date/Time to String" on page 336

• "time() — Determine Current Time" on page 373
```

div() — Calculate Quotient and Remainder

• "<time.h>" on page 17

Format

```
#include <stdlib.h>
div_t div(int numerator, int denominator);
```

Language Level: ANSI

Thread Safe: YES. However, only the function version is thread safe. The macro version is NOT thread safe.

Description

The div() function calculates the quotient and remainder of the division of *numerator* by *denominator*.

Return Value

The div() function returns a structure of type div_t, containing both the quotient int quot and the remainder int rem. If the return value cannot be represented, its value is undefined. If *denominator* is 0, an exception will be raised.

Example that uses div()

This example uses div() to calculate the quotients and remainders for a set of two dividends and two divisors.

```
#include <stdlib.h>
#include <stdio.h>
int main(void)
   int num[2] = \{45, -45\};
   int den[2] = \{7,-7\};
   div_t ans; /* div_t is a struct type containing two ints:
                     'quot' stores quotient; 'rem' stores remainder */
   short i,j;
   printf("Results of division:\n");
   for (i = 0; i < 2; i++)
      for (j = 0; j < 2; j++)
          ans = div(num[i],den[j]);
          printf("Dividend: %6d Divisor: %6d", num[i], den[j]);
          printf(" Quotient: %6d Remainder: %6d\n", ans.quot, ans.rem);
/****** Output should be similar to: *********
Results of division:
Dividend: 45 Divisor: 7 Quotient: 6 Remainder: 3
Dividend: 45 Divisor: -7 Quotient: -6 Remainder: 3
Dividend: -45 Divisor: 7 Quotient: -6 Remainder: -3
Dividend: -45 Divisor: -7 Quotient: 6 Remainder: -3
```

Related Information

- "ldiv() lldiv() Perform Long and Long Long Division" on page 156
- "<stdlib.h>" on page 16

erf() - erfc() - Calculate Error Functions

Format

```
#include <math.h>
double erf(double x);
double erfc(double x);
```

Language Level: ILE C Extension

Description

The erf() function calculates the error function of:

$$2\pi^{-1/2}\int_{0}^{x}e^{-t^{2}}dt$$

The erfc() function computes the value of 1.0 - erf(x). The erfc() function is used in place of erf() for large values of x.

Return Value

The erf() function returns a double value that represents the error function. The erfc() function returns a double value representing 1.0 - erf.

Example that uses erf()

This example uses erf() and erfc() to compute the error function of two numbers.

```
#include <stdio.h>
#include <math.h>
double smallx, largex, value;
int main(void)
  smallx = 0.1;
  largex = 10.0;
  value = erf(smallx);
                         /* value = 0.112463 */
  printf("Error value for 0.1: %lf\n", value);
  value = erfc(largex);
                             /* value = 2.088488e-45 */
  printf("Error value for 10.0: %le\n", value);
/******* Output should be similar to: *********
Error value for 0.1: 0.112463
Error value for 10.0: 2.088488e-45
```

Related Information

- "Bessel Functions" on page 51
- "gamma() Gamma Function" on page 131
- "<math.h>" on page 8

exit() — End Program

Format

```
#include <stdlib.h>
void exit(int status);
```

Language Level: ANSI

Thread Safe: YES.

Description

The exit() function returns control to the host environment from the program. It first calls all functions that are registered with the atexit() function, in reverse order; that is, the last one that is registered is the first one called. It deletes all buffers and closes all open files before ending the program.

The argument *status* can have a value from 0 to 255 inclusive, or be one of the macros EXIT_SUCCESS or EXIT_FAILURE. A *status* value of EXIT_SUCCESS or 0 indicates a normal exit; otherwise, another status value is returned.

Note: When compiled with SYSIFCOPT(*ASYNCSIGNAL), exit() cannot be called in a signal handler.

Return Value

The exit() function returns both control and the value of *status* to the operating system.

Example that uses exit()

This example ends the program after deleting buffers and closing any open files if it cannot open the file myfile.

```
#include <stdio.h>
#include <stdlib.h>

FILE *stream;

int main(void)
{
    if ((stream = fopen("mylib/myfile", "r")) == NULL)
    {
        perror("Could not open data file");
        exit(EXIT_FAILURE);
    }
}
```

Related Information

- "abort() Stop a Program" on page 37
- "atexit() Record Program Ending Function" on page 46
- "signal() Handle Interrupt Signals" on page 313
- "<stdlib.h>" on page 16

exp() — Calculate Exponential Function

Format

```
#include <math.h>
double exp(double x);
```

Language Level: ANSI

Description

The exp() function calculates the exponential value of a floating-point argument x (e^{x} , where *e* equals 2.17128128...).

Return Value

If an overflow occurs, the exp() function returns HUGE_VAL. If an underflow occurs, it returns 0. Both overflow and underflow set errno to ERANGE. The value of errno may also be set to EDOM.

Example that uses exp()

This example calculates y as the exponential function of x:

```
#include <math.h>
#include <stdio.h>
int main(void)
  double x, y;
  x = 5.0;
  y = exp(x);
  printf("exp(%lf) = %lf\n", x, y);
/******* Output should be similar to: *********
exp(5.000000) = 148.413159
```

Related Information

- "log() Calculate Natural Logarithm" on page 165
- "log10() Calculate Base 10 Logarithm" on page 166
- "<math.h>" on page 8

fabs() — Calculate Floating-Point Absolute Value

Format

```
#include <math.h>
double fabs(double x);
```

Language Level: ANSI

Description

The fabs() function calculates the absolute value of the floating-point argument x.

Return Value

The fabs() function returns the absolute value. There is no error return value.

Example that uses fabs()

This example calculates *y* as the absolute value of *x*:

Related Information

- "abs() Calculate Integer Absolute Value" on page 38
- "labs() llabs() Calculate Absolute Value of Long and Long Long Integer" on page 155
- "<math.h>" on page 8

fclose() — Close Stream

Format

```
#include <stdio.h>
int fclose(FILE *stream);
```

Language Level: ANSI

Thread Safe: YES

Description

The fclose() function closes a stream pointed to by *stream*. This function deletes all buffers that are associated with the stream before closing it. When it closes the stream, the function releases any buffers that the system reserved. When a binary stream is closed, the last record in the file is padded with null characters (\0) to the end of the record.

Return Value

The fclose() function returns 0 if it successfully closes the stream, or EOF if any errors were detected.

The value of errno may be set to:

Value Meaning

ENOTOPEN

The file is not open.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

Note: The storage pointed to by the FILE pointer is freed by the fclose() function. After the use of the fclose() function, any attempt to use the FILE pointer is not valid.

Example that uses fclose()

#include <stdio.h>

This example opens a file myfile for reading as a stream; then it closes this file.

```
#define NUM ALPHA 26
int main(void)
 FILE *stream;
 char buffer[NUM_ALPHA];
 if (( stream = fopen("mylib/myfile", "r"))!= NULL )
    fread( buffer, sizeof( char ), NUM ALPHA, stream );
   printf( "buffer = %s\n", buffer );
 if (fclose(stream)) /* Close the stream. */
     perror("fclose error");
 else printf("File mylib/myfile closed successfully.\n");
```

Related Information

- "fflush() Write Buffer to File" on page 80
- "fopen() Open Files" on page 92
- "freopen() Redirect Open Files" on page 111
- "<stdio.h>" on page 15

fdopen() — Associates Stream With File Descriptor

Format

```
#include <stdio.h>
FILE *fdopen(int handle, char *type);
```

Language Level: XPG4

Thread Safe: YES

Description

The fdopen() function is made available by specifying SYSIFCOPT(*IFSIO) on the compilation command.

The fdopen() function associates an input or output stream with the file that is identified by handle. The type variable is a character string specifying the type of access that is requested for the stream.

Mode Description

1

- r Create a stream to read a text file. The file pointer is set to the beginning of the file.
- **w** Create a stream to write to a text file. The file pointer is set to the beginning of the file.
- a Create a stream to write, in append mode, at the end of the text file. The file pointer is set to the end of the file.
- **r+** Create a stream for reading and writing a text file. The file pointer is set to the beginning of the file.
- **w+** Create a stream for reading and writing a text file. The file pointer is set to the beginning of the file.
- **a+** Create a stream for reading or writing, in append mode, at the end of the text file. The file pointer is set to the end of the file.
- **rb** Create a stream to read a binary file. The file pointer is set to the beginning of the file.
- **wb** Create a stream to write to a binary file. The file pointer is set to the beginning of the file.
- **ab** Create a stream to write to a binary file in append mode. The file pointer is set to the end of the file.

r+b or rb+

Create a stream for reading and writing a binary file. The file pointer is set to the beginning of the file.

w+b or wb+

Create a stream for reading and writing a binary file. The file pointer is set to the beginning of the file.

a+b or ab+

Create a stream for reading and writing to a binary file in append mode. The file pointer is set to the end of the file.

Note: Use the w, w+, wb, wb+, and w+b modes with care; they can destroy existing files.

The specified *type* must be compatible with the access method you used to open the file. If the file was opened with the O_APPEND flag, the stream mode must be a, a+, ab, a+b, or ab+. To use the fdopen() function you need a file descriptor. To get a descriptor use the POSIX function open(). The O_APPEND flag is a mode for open(). Modes for open() are defined in QSYSINC/H/FCNTL. For further information see the API topic.

If fdopen() returns NULL, use close() to close the file. If fdopen() is successful, you must use fclose() to close the stream and file.

Return Value

The fdopen() function returns a pointer to a file structure that can be used to access the open file. A NULL pointer return value indicates an error.

Example that uses fdopen()

This example opens the file sample.dat and associates a stream with the file using fdopen(). It then reads from the stream into the buffer.

```
/* compile with SYSIFCOPT(*IFSIO) */
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
int main(void)
  long length;
  int fh;
  char buffer[20];
  FILE *fp;
  printf("\nCreating sample.dat.\n");
  if ((fp= fopen("/sample.dat", "w")) == NULL) {
      perror(" File was not created: ");
      exit(1);
  fputs("Sample Program", fp);
  fclose(fp);
  memset(buffer, '\0', 20);
                                                        /* Initialize buffer*/
  if (-1 == (fh = open("/sample.dat", O_RDWR|O_APPEND))) {
     perror("Unable to open sample.dat");
     exit(1);
  if (NULL == (fp = fdopen(fh, "r"))) {
     perror("fdopen failed");
     close(fh);
     exit(1);
  if (14 != fread(buffer, 1, 14, fp)) {
     perror("fread failed");
     fclose(fp);
     exit(1);
  printf("Successfully read from the stream the following:\n%s.\n", buffer);
  fclose(fp);
  return 1;
  /************************
   * The output should be:
   * Creating sample.dat.
   * Successfully read from the stream the following:
   * Sample Program.
Related Information
```

- "fclose() Close Stream" on page 75
- "fopen() Open Files" on page 92
- "fseek() fseeko() Reposition File Position" on page 114
- "fsetpos() Set File Position" on page 117
- "rewind() Adjust Current File Position" on page 245
- "<stdio.h>" on page 15
- open See the API topic
- close See the API topic

feof() — Test End-of-File Indicator

Format

```
#include <stdio.h>
int feof(FILE *stream);
```

Language Level: ANSI

Description

The feof() function indicates whether the end-of-file flag is set for the given *stream*. The end-of-file flag is set by several functions to indicate the end of the file. The end-of-file flag is cleared by calling the rewind(), fsetpos(), fseek(), or clearerr() functions for this stream.

Return Value

The feof() function returns a nonzero value if and only if the EOF flag is set; otherwise, it returns 0.

Example that uses feof()

This example scans the input stream until it reads an end-of-file character.

```
#include <stdio.h>
#include <stdib.h>

int main(void)
{
    char string[100];
    FILE *stream;
    memset(string, 0, sizeof(string));
    stream = fopen("qcpple/qacsrc(feof)", "r");

    fscanf(stream, "%s", string);
    while (!feof(stream))
    {
        printf("%s\n", string);
        memset(string, 0, sizeof(string));
        fscanf(stream, "%s", string);
    }
}
```

Related Information

- "clearerr() Reset Error Indicators" on page 62
- "ferror() Test for Read/Write Errors"
- "fseek() fseeko() Reposition File Position" on page 114
- "fsetpos() Set File Position" on page 117
- "perror() Print Error Message" on page 198
- "rewind() Adjust Current File Position" on page 245
- "<stdio.h>" on page 15

ferror() — Test for Read/Write Errors

Format

```
#include <stdio.h>
int ferror(FILE *stream);
```

Language Level: ANSI

Description

The ferror() function tests for an error in reading from or writing to the given stream. If an error occurs, the error indicator for the stream remains set until you close *stream*, call the rewind() function, or call the clearerr() function.

Return Value

The ferror() function returns a nonzero value to indicate an error on the given stream. A return value of 0 means that no error has occurred.

Example that uses ferror()

This example puts data out to a stream, and then checks that a write error has not occurred.

```
#include <stdio.h>
int main(void)
  FILE *stream;
  char *string = "Important information";
  stream = fopen("mylib/myfile", "w");
   fprintf(stream, "%s\n", string);
   if (ferror(stream))
      printf("write error\n");
      clearerr(stream);
   if (fclose(stream))
      perror("fclose error");
```

Related Information

- "clearerr() Reset Error Indicators" on page 62
- "feof() Test End-of-File Indicator" on page 79
- "fopen() Open Files" on page 92
- "perror() Print Error Message" on page 198
- "strerror() Set Pointer to Run-Time Error Message" on page 333
- "<stdio.h>" on page 15

fflush() — Write Buffer to File

Format

```
#include <stdio.h>
int fflush(FILE *stream);
```

Language Level: ANSI

Thread Safe: YES

Description

The fflush() function causes the system to empty the buffer that is associated with the specified output stream, if possible. If the stream is open for input, the fflush() function undoes the effect of any ungetc() function. The stream remains open after the call.

If stream is NULL, the system flushes all open streams.

Note: The system automatically deletes buffers when you close the stream, or when a program ends normally without closing the stream.

Return Value

The fflush() function returns the value 0 if it successfully deletes the buffer. It returns EOF if an error occurs.

The value of errno may be set to:

Value Meaning

ENOTOPEN

The file is not open.

ERECIO

The file is opened for record I/O.

ESTDIN

stdin cannot be opened.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

The fflush() function is not supported for files that are opened with type=record.

Example that uses fflush()

This example deletes a stream buffer.

```
#include <stdio.h>
int main(void)
  FILE *stream:
  int ch;
  unsigned int result = 0;
  stream = fopen("mylib/myfile", "r");
  while ((ch = getc(stream)) != EOF && isdigit(ch))
     result = result * 10 + ch - '0';
   if (ch != EOF)
     ungetc(ch, stream);
  fflush(stream);
                             /* fflush undoes the effect of ungetc function
   printf("The result is: %d\n", result);
  if ((ch = getc(stream)) != EOF)
     printf("The character is: %c\n", ch);
```

Related Information

- "fclose() Close Stream" on page 75
- "fopen() Open Files" on page 92
- "setbuf() Control Buffering" on page 305
- "ungetc() Push Character onto Input Stream" on page 381
- "<stdio.h>" on page 15

fgetc() — Read a Character

Format

```
#include <stdio.h>
int fgetc(FILE *stream);
```

Language Level: ANSI

Thread Safe: YES

Description

The fgetc() function reads a single unsigned character from the input stream at the current position and increases the associated file pointer, if any, so that it points to the next character.

Note: The fgetc()function is identical to getc(), but it is always defined as a function call; it is never replaced by a macro.

Return Value

The fgetc() function returns the character that is read as an integer. An EOF return value indicates an error or an end-of-file condition. Use the feof() or the ferror() function to determine whether the EOF value indicates an error or the end of the file.

The value of errno may be set to:

Value Meaning

EBADF

1

The file pointer or descriptor is not valid.

ENOTREAD

The file is not open for read operations.

EGETANDPUT

An read operation that was not allowed occurred after a write operation.

ERECIO

The file is open for record I/O.

ESTDIN

stdin cannot be opened.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

The fgetc() function is not supported for files that are opened with type=record.

Example that uses fgetc()

This example gathers a line of input from a stream.

```
#include <stdio.h>
#define MAX LEN 80
int main(void)
  FILE *stream;
  char buffer[MAX_LEN + 1];
  int i, ch;
  stream = fopen("mylib/myfile","r");
  for (i = 0; (i < (sizeof(buffer)-1) &&
       ((ch = fgetc(stream)) != EOF) && (ch != '\n')); i++)
    buffer[i] = ch;
  buffer[i] = '\0';
  if (fclose(stream))
    perror("fclose error");
  printf("line: %s\n", buffer);
/***********************************
    If FILENAME contains: one two three
    The output should be:
    line: one two three
```

Related Information

- "feof() Test End-of-File Indicator" on page 79
- "ferror() Test for Read/Write Errors" on page 79
- "fgetwc() Read Wide Character from Stream" on page 86
- "fputc() Write Character" on page 101
- "getc() getchar() Read a Character" on page 133

- "getwc() Read Wide Character from Stream" on page 138
- "getwchar() Get Wide Character from stdin" on page 140
- "<stdio.h>" on page 15

fgetpos() — Get File Position

Format

```
#include <stdio.h>
int fgetpos(FILE *stream, fpos t *pos);
```

Language Level: ANSI

Thread Safe: YES

Description

The fgetpos() function stores the current position of the file pointer that is associated with stream into the object pointed to by pos. The value pointed to by pos can be used later in a call to fsetpos() to reposition the stream.

Return Value

The fgetpos() function returns 0 if successful; on error, it returns nonzero and sets errno to a nonzero value.

The value of errno may be set to:

Value Meaning

EBADF

The file pointer or descriptor is not valid.

EBADSEEK

Bad offset for a seek operation.

ENODEV

Operation was attempted on a wrong device.

ENOTOPEN

The file is not open.

ERECIO

The file is open for record I/O.

ESTDERR

stderr cannot be opened.

ESTDIN

stdin cannot be opened.

ESTDOUT

stdout cannot be opened.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

The fgetpos() function is not supported for files that are opened with type=record.

Example that uses fgetpos()

This example opens the file myfile for reading and stores the current file pointer position into the variable *pos*.

Related Information

- "fseek() fseeko() Reposition File Position" on page 114
- "fsetpos() Set File Position" on page 117
- "ftell() ftello() Get Current Position" on page 118
- "<stdio.h>" on page 15

fgets() — Read a String

I

Format

```
#include <stdio.h>
char *fgets (char *string, int n, FILE *stream);
```

Language Level: ANSI

Thread Safe: YES

Description

The fgets() function reads characters from the current *stream* position up to and including the first new-line character (\n), up to the end of the stream, or until the number of characters read is equal to n-1, whichever comes first. The fgets() function stores the result in *string* and adds a null character (\n 0) to the end of the string. The *string* includes the new-line character, if read. If n is equal to 1, the *string* is empty.

The value of errno can be set to **EBADF**,(the catalog descriptor is not valid).

Return Value

The fgets() function returns a pointer to the *string* buffer if successful. A NULL return value indicates an error or an end-of-file condition. Use the feof() or ferror() functions to determine whether the NULL value indicates an error or the end of the file. In either case, the value of the string is unchanged.

The fgets() function is not supported for files that are opened with type=record.

Example that uses fgets()

This example gets a line of input from a data stream. The example reads no more than MAX_LEN - 1 characters, or up to a new-line character from the stream.

```
#include <stdio.h>
#define MAX LEN 100
int main(void)
  FILE *stream;
  char line[MAX_LEN], *result;
  stream = fopen("mylib/myfile","rb");
  if ((result = fgets(line,MAX LEN,stream)) != NULL)
       printf("The string is \$s \ n", result);
   if (fclose(stream))
       perror("fclose error");
```

Related Information

- "feof() Test End-of-File Indicator" on page 79
- "ferror() Test for Read/Write Errors" on page 79
- "fgetws() Read Wide-Character String from Stream" on page 88
- "fputs() Write String" on page 103
- "gets() Read a Line" on page 137
- "puts() Write a String" on page 212
- "<stdio.h>" on page 15

fgetwc() — Read Wide Character from Stream

Format

```
#include <wchar.h>
#include <stdio.h>
wint_t fgetwc(FILE *stream);
```

Language Level: ANSI

Thread Safe: YES

Description

The fgetwc() reads the next multibyte character from the input stream pointed to by stream, converts it to a wide character, and advances the associated file position indicator for the stream (if defined).

Using non-wide-character functions with fgetwc() on the same stream results in undefined behavior. After calling fgetwc(), flush the buffer or reposition the stream pointer before calling a write function for the stream, unless EOF has been reached. After a write operation on the stream, flush the buffer or reposition the stream pointer before calling fgetwc().

Note:

The behavior of the fgetwc() function is affected by the LC_CTYPE category of the current locale. If you change the category between subsequent read operations on the same stream, undefined results can occur.

This function is available when SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) are specified on the compilation command.

Return Value

The fgetwc() function returns the next wide character that corresponds to the multibyte character from the input stream pointed to by *stream*. If the stream is at EOF, the EOF indicator for the stream is set, and fgetwc() returns WEOF.

If a read error occurs, the error indicator for the stream is set, and the fgetwc() function returns WEOF. If an encoding error occurs (an error converting the multibyte character into a wide character), the fgetwc() function sets error to EILSEQ and returns WEOF.

Use the ferror() and feof() functions to distinguish between a read error and an EOF. EOF is only reached when an attempt is made to read past the last byte of data. Reading up to and including the last byte of data does not turn on the EOF indicator.

Example that uses fgetwc()

This example opens a file, reads in each wide character, and prints out the characters.

```
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <errno.h>
int main(void)
  FILE *stream;
  wint t wc;
   if (NULL == (stream = fopen("fgetwc.dat", "r"))) {
     printf("Unable to open: \"fgetwc.dat\"\n");
     exit(1);
   }
  errno = 0;
  while (WEOF != (wc = fgetwc(stream)))
     printf("wc = %lc\n", wc);
  if (EILSEQ == errno) {
     printf("An invalid wide character was encountered.\n");
     exit(1);
  fclose(stream);
  return 0;
 * * End of File * * *
```

Related Information

- "fgetc() Read a Character" on page 82
- "fputwc() Write Wide Character" on page 104
- "fgetws() Read Wide-Character String from Stream"
- "getc() getchar() Read a Character" on page 133
- "getwc() Read Wide Character from Stream" on page 138
- "getwchar() Get Wide Character from stdin" on page 140
- "<stdio.h>" on page 15
- "<wchar.h>" on page 18

fgetws() — Read Wide-Character String from Stream

Format

```
#include <wchar.h>
#include <stdio.h>
wchar t *fgetws(wchar t *wcs, int n, FILE *stream);
```

Language Level: ANSI

Thread Safe: YES

Description

The fgetws() function reads at most one less than the number of wide characters specified by n from the stream pointed to by stream. The fgetws() function stops reading characters after WEOF, or after it reads a new-line wide character (which is retained). It adds a null wide character immediately after the last wide character read into the array. The fgetws() function advances the file position unless there is an error. If an error occurs, the file position is undefined.

Using non-wide-character functions with the fgetws () function on the same stream results in undefined behavior. After calling the fgetws() function, flush the buffer or reposition the stream pointer before calling a write function for the stream, unless WEOF has been reached. After a write operation on the stream, flush the buffer or reposition the stream pointer before calling the fgetws () function.

Note: The behavior of the fgetws() function is affected by the LC_CTYPE category of the current locale. If you change the category between subsequent read operations on the same stream, undefined results can occur.

Return Value

If successful, the fqetws() function returns a pointer to the wide-character string wcs. If WEOF is encountered before any wide characters have been read into wcs, the contents of wcs remain unchanged and the fgetws() function returns a null pointer. If WEOF is reached after data has already been read into the string buffer, the fgetws() function returns a pointer to the string buffer to indicate success. A subsequent call would return NULL because WEOF would be reached without any data being read.

If a read error occurs, the contents of wcs are indeterminate, and the fgetws() function returns NULL. If an encoding error occurs (in converting a wide character to a multibyte character), the fgetws() function sets errno to EILSEQ and returns NULL.

If n equals 1, the wcs buffer has only room for the ending null character, and nothing is read from the stream. (Such an operation is still considered a read operation, so it cannot immediately follow a write operation unless the buffer is flushed or the stream pointer repositioned first.) If n is greater than 1, the fgetws () function fails only if an I/O error occurs, or if WEOF is reached before data is read from the stream.

Use the ferror() and feof() functions to distinguish between a read error and a WEOF. A WEOF error is only reached when an attempt is made to read past the last byte of data. Reading up to and including the last byte of data does not turn on the WEOF indicator.

Note: This function is available when SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) are specified on the compilation command.

Example that uses fgetws()

This example opens a file, reads in the file contents, then prints the file contents.

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
int main(void)
  FILE *stream:
  wchar t wcs[100];
  if (NULL == (stream = fopen("fgetws.dat", "r"))) {
     printf("Unable to open: \"fgetws.dat\"\n");
     exit(1);
  errno = 0;
  if (NULL == fgetws(wcs, 100, stream)) {
     if (EILSEQ == errno) {
       printf("An invalid wide character was encountered.\n");
       exit(1);
     else if (feof(stream))
            printf("End of file reached.\n");
            perror("Read error.\n");
  printf("wcs = \"%ls\"\n", wcs);
  fclose(stream);
  return 0;
  /***********************
     Assuming the file fgetws.dat contains:
     This test string should not return -1
     The output should be similar to:
     wcs = "This test string should not return -1"
```

Related Information

- "fgetc() Read a Character" on page 82
- "fgets() Read a String" on page 85
- "fgetwc() Read Wide Character from Stream" on page 86
- "fputws() Write Wide-Character String" on page 106
- "<stdio.h>" on page 15
- "<wchar.h>" on page 18

fileno() — Determine File Handle

Format

```
#include <stdio.h>
int fileno(FILE *stream);
```

Language Level: XPG4

Description

The fileno() function only works in ILE C when the Integrated File System has been enabled. The fileno() function determines the file handle that is currently associated with stream.

The value of errno can be set to EBADF.

Return Value

If the environment variable QIBM_USE_DESCRIPTOR_STDIO is set to Yes, the fileno()function returns 0 for stdin, 1 for stdout, and 2 for stderr.

With QIBM_USE_DESCRIPTOR_STDIO set to No, the ILE C session files stdin, stdout, and stderr do not have a file descriptor associated with them. The fileno() function will return a value of -1 in this case.

Example that uses fileno()

This example determines the file handle of the stderr data stream.

```
/* Compile with SYSIFCOPT(*IFSIO)
#include <stdio.h>
int main (void)
  FILE *fp;
  int result;
  fp = fopen ("stderr", "w");
  result = fileno(fp);
  printf("The file handle associated with stderr is %d.\n", result);
  return 0;
  /***********************************
   * The output should be:
   * The file handle associated with stderr is -1.
```

Related Information

- "fopen() Open Files" on page 92
- "freopen() Redirect Open Files" on page 111
- "<stdio.h>" on page 15

floor() —Find Integer <=Argument

Format

```
#include <math.h>
double floor(double x);
```

Language Level: ANSI

Description

The floor() function calculates the largest integer that is less than or equal to x.

Return Value

The floor() function returns the floating-point result as a double value.

The result of floor() cannot have a range error.

Example that uses floor()

This example assigns y value of the largest integer less than or equal to 2.8 and z the value of the largest integer less than or equal to -2.8.

Related Information

- "ceil() Find Integer >= Argument" on page 61
- "fmod() Calculate Floating-Point Remainder"
- "<math.h>" on page 8

fmod() — Calculate Floating-Point Remainder

Format

```
#include <math.h>
double fmod(double x, double y);
```

Language Level: ANSI

Description

The fmod() function calculates the floating-point remainder of x/y. The absolute value of the result is always less than the absolute value of y. The result will have the same sign as x.

Return Value

The fmod() function returns the floating-point remainder of x/y. If y is zero or if x/y causes an overflow, fmod() returns 0. The value of errno may be set to EDOM.

Example that uses fmod()

This example computes z as the remainder of x/y; here, x/y is -3 with a remainder of -1.

```
#include <math.h>
#include <stdio.h>
int main(void)
  double x, y, z;
  x = -10.0;
  y = 3.0;
                   /* z = -1.0 */
  z = fmod(x,y);
  printf("fmod(%lf, %lf) = %lf\n", x, y, z);
/******* Output should be similar to: *********
fmod(-10.000000, 3.000000) = -1.000000
```

Related Information

- "ceil() Find Integer >= Argument" on page 61
- "fabs() Calculate Floating-Point Absolute Value" on page 74
- "floor() —Find Integer <= Argument" on page 91
- "<math.h>" on page 8

fopen() — Open Files

Format

```
#include <stdio.h>
FILE *fopen(const char *filename, const char *mode);
```

Language Level: ANSI

Thread Safe: YES

Description

The fopen() function opens the file that is specified by *filename*. The *mode* parameter is a character string specifying the type of access that is requested for the file. The mode variable contains one positional parameter followed by optional keyword parameters.

Note: Because the fopen() API calls the open() API, the owner of the file is given read, write, and execute authority to the file.

The possible values for the positional parameters are:

Mode Description

| |

- r Open a text file for reading. The file must exist.
- **w** Create a text file for writing. If the given file exists, its contents are destroyed unless it is a logical file.
- a Open a text file in append mode for writing at the end of the file. The fopen() function creates the file if it does not exist and is not a logical file.
- r+ Open a text file for both reading and writing. The file must exist.
- **w+** Create a text file for both reading and writing. If the given file exists, its contents are cleared unless it is a logical file.
- a+ Open a text file in append mode for reading or updating at the end of the file. The fopen() function creates the file if it does not exist.
- **rb** Open a binary file for reading. The file must exist.
- **wb** Create an empty binary file for writing. If the file exists, its contents are cleared unless it is a logical file.
- **ab** Open a binary file in append mode for writing at the end of the file. The fopen function creates the file if it does not exist.

r+b or rb+

Open a binary file for both reading and writing. The file must exist.

w+b or wb+

Create an empty binary file for both reading and writing. If the file exists, its contents will be cleared unless it is a logical file.

a+b or ab+

Open a binary file in append mode for writing at the end of the file. The fopen() function creates the file if it does not exist.

The fopen() function is not supported for files that are opened with the attributes type=record and ab+, rb+, or wb+.

Note: Use the w, w+, wb, w+b, and wb+ parameters with care; data in existing files of the same name will be lost.

Text files contain printable characters and control characters that are organized into lines. Each line ends with a new-line character, except possibly the last line, depending on the compiler. The system may insert or convert control characters in an output text stream. The fopen() function mode "a" and "a+" can not be used for the QSYS.LIB file system. There are implementation restrictions when using the QSYS.LIB file system for text files in all modes. Seeking beyond the start of files cannot be relied on to work with streams opened in text mode.

If a binary file does not exist, you can create one using the following command:

CRTSRCPF FILE(MYLIB/MYFILE) RCDLEN(LRECL) MBR(MYMBR) SYSTEM(*FILETYPE)

Note: Data output to a text stream may not compare as equal to the same data on input. The QSYS.LIB file system treats database files as a directory of

The Integrated File System supports files of up to 4GB in size. There are three methods you can use to allow your application programs access to 64-bit C runtime functions for IFS files larger than 2 gigabytes:

- · Specify SYSIFCOPT(*IFS64IO) on a compilation command, which causes the native C compiler to define _IFS64_IO_. This causes the macros _LARGE_FILES and _LARGE_FILE_API to be defined.
- Define the macro _LARGE_FILES, either in the program source or by specifying DEFINE('_LARGE_FILES') on a compilation command. The existing C runtime functions and the relevant data types in the code will all be automatically mapped or redefined to their 64-bit versions.
- Define the macro LARGE_FILE_API, either in the program source or by specifying DEFINE('_LARGE_FILE_API') on a compilation command. This makes visible the set of of new 64-bit C runtime functions and data types. The application must explicitly specify the name of the C runtime functions, both existing version and 64-bit version, to use.

The 64-bit C runtime functions include the following: int fgetpos64(), FILE *fopen64(), FILE *freopen64(), FILE *wfopen64(), int fsetpos64(FILE *, const fpost64 t *), FILE *tmpfile64(), int fseeko(FILE *, off t, int), int fseeko64(FILE *, off64 t, int), off t ftello(FILE *), and off64 t ftello64().

Binary files contain a series of characters. For binary files, the system does not translate control characters on input or output.

If a text file does not exist, you can create one using the following command:

CRTPF FILE(MYLIB/MYFILE) RCDLEN(LRECL) MBR(MYMBR) MAXMBRS(*NOMAX) SYSTEM(*FILETYPE)

When you open a file with a, a+, ab, a+b or ab+ mode, all write operations take place at the end of the file. Although you can reposition the file pointer using the fseek() function or the rewind() function, the write functions move the file pointer back to the end of the file before they carry out any operation. This action prevents you from overwriting existing data.

When you specify the update mode (using + in the second or third position), you can both read from and write to the file. However, when switching between reading and writing, you must include an intervening positioning function such as the fseek(), fsetpos(), rewind(), or fflush(). Output may immediately follow input if the end-of-file was detected.

Keyword parameters for non-Integrated File System

blksize=value

Specifies the maximum length, in bytes, of a physical block of records.

lrecl=value

Specifies the length, in bytes, for fixed-length records and the maximum length for variable-length records.

recfm=value

value can be:

F fixed-length, deblocked records

FB fixed-length, blocked records

V variable-length, deblocked records

VB variable-length, blocked records

VBS variable-length, blocked, spanned records for tape files

VS variable-length, deblocked, spanned records for tape files

D variable-length, deblocked, unspanned records for ASCII D format for tape files

DB variable-length, blocked, unspanned records for ASCII D format for tape files

U undefined format for tape files

FA fixed-length that uses first character forms control data for printer files

Note: If the file is created using CTLCHAR(*FCFC), the first character form control will be used. If it is created using CTLCHAR(*NONE), the first character form control will not be used.

commit=value

value can be:

N This parameter identifies that this file is not opened under commitment control. This is the default.

Y This parameter identifies that this file is opened under commitment control.

ccsid=value

If a CCSID that is not supported by the iSeries is specified, it is ignored by data management. The default is the job's CCSID value.

arrseq=value

value can be:

N This parameter identifies that this file is processed in the way it was created. This is the default.

Y This parameter identifies that this file is processed in arrival sequence.

indicators=value

value can be:

N This parameter identifies that indicators in display, ICF, or printer files are stored in the file buffer. This is the default.

Y This parameter identifies that indicators in display, ICF, or printer files are stored in a separate indicator area, not in the file buffer. A file buffer is the area the iSeries system uses to transfer data to and from the user program and the operating system when writing and reading. You must store indicators in a separate indicator area when processing ICF files.

type=value

value can be:

memory This parameter identifies this file as a memory file that is accessible only from C programs. This is the default.

record This parameter specifies that the file is to be opened for sequential record I/O. The file must be opened as a binary file; otherwise, the

Keyword parameters for Integrated File System only

type=value

value can be:

record The file is opened for sequential record I/O. (File has to be opened as binary stream.)

ccsid=value

ccsid is converted to a code page value. The default is to use the job CCSID value as the code page. The CCSID and codepage option cannot both be specified. The CCSID option provides compatibility with iSeries and Data management based stream I/O.

Note: Mixed data (the data contains both single and double-byte characters) is not supported for a file data processing mode of text. Mixed data is supported for a file processing mode of binary.

If you specify the ccsid keyword, you cannot specify the o_ccsid keyword or the codepage keyword.

Because of the possible expansion or contraction of converted data, making assumptions about data size and the current file offset is dangerous. For example, a file might have a physical size of 100 bytes, but after an application has read 100 bytes from the file, the current file offset may be only 50. In order to read the whole file, the application might have to read 200 bytes or more, depending on the CCSIDs involved. Therefore, file positioning functions such as ftell(), fseek(), fgetpos(), and fsetpos() will not work. These functions will fail with ENOTSUP. Read functions also will not work if buffering is on, as it is by default. To turn buffering off, use the setvbuf function with the _IONBF keyword.

The fopen() function will fail with the ECONVERT error when all of the following three conditions occur:

- The file data processing mode is text.
- The code page is not specified.
- The CCSID of the job is 'mixed-data' (the data contains both single– and double–byte characters).

o ccsid=value

This parameter is similar to the *ccsid* parameter, except that the value specified is not converted to a code page. Also, mixed data is supported. If the file is created, it is tagged with the specified CCSID. If the file already exists, data will be converted from the CCSID of the file to the specified CCSID on read operations. On write operations, the data is assumed to be in the specified CCSID, and is converted to the CCSID of the file.

Because of the possible expansion or contraction of converted data, making assumptions about data size and the current file offset is dangerous. For example, a file might have a physical size of 100 bytes, but after an application has read 100 bytes from the file, the current file offset may be only 50. In order to read the whole file, the application might have to read 200 bytes or more, depending on the CCSIDs involved. Therefore, file positioning functions such as ftell(), fseek(), fgetpos(), and fsetpos()

will not work. These functions will fail with ENOTSUP. Read functions also will not work if buffering is on, as it is by default. To turn buffering off, use the setvbuf function with the _IONBF keyword.

Example that uses *o_ccsid*

| |

1

```
/* Create a file that is tagged with CCSID 37 */
if ((fp = fopen("/MYFILE" , "w, o_ccsid=37")) == NULL) {
   printf("Failed to open file with o_ccsid=37\n");
fclose(fp);
/* Now reopen the file with CCSID 13488, because your application
 wants to deal with the data in UNICODE */
if ((fp = fopen("/MYFILE", "r+, o_ccsid=13488")) == NULL) {
   printf("Failed to open file with o ccsid=13488\n");
/* Turn buffering off because read functions do not work when
buffering is on */
if (setbuf(fp, NULL, _IONBF, 0) != 0){
    printf("Unable to turn buffering off\n");
/* Because you opened with o ccsid = 13488, you must provide
all input data as unicode.
If this program is compiled with LOCALETYPE(*LOCALEUCS2),
L constrants will be unicode. */
funcreturn = fputws(L"ABC", fp); /* Write a unicode ABC to the file. */
if (funcreturn < 0) {
   printf("Error with 'fputws' on line %d\n", LINE );
/* Because the file was tagged with CCSID 37, the unicode ABC was
converted to EBCDIC ABC when it was written to the file. */
```

codepage=value

The code page that is specified by value is used.

If you specify the codepage keyword, you cannot specify the ccsid keyword or the o_ccsid keyword.

If the file to be opened does not exist, and the open mode specifies that the file should be created, the file is created and tagged with the calculated code page. If the file already exists, the data read from the file is converted from the files code page to the calculated code page during the read operation. Data written to the file is assumed to be in the calculated code page and is converted to the code page of the file during the write operation.

crln=value

value can be:

Y The line terminator to be used is carriage return [CR], new line [NL] combination. When data is read, all carriage returns [CR] are stripped for string functions. When data is written to a file, carriage returns [CR] are added before each new line [NL] character. Line terminator processing only occurs when a file is open with text mode. This is the default.

N The line terminator to be used is new line [NL] only.

The keyword parameters are not case sensitive and should be separated by a comma.

The fopen() function generally fails if parameters are mismatched.

Return Value

The fopen() function returns a pointer to a FILE structure type that can be used to access the open file.

Note: To use stream files (type = record) with record I/O functions, you must cast the FILE pointer to an RFILE pointer.

A NULL pointer return value indicates an error.

The value of errno may be set to:

Value Meaning

EBADMODE

The file mode that is specified is not valid.

EBADNAME

The file name that is specified is not valid.

ECONVRT

Conversion error.

ENOENT

No file or library.

ENOMEM

Storage allocation request failed.

ENOTOPEN

The file is not open.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

If the mode string passed to fopen() is correct, fopen() will not set errno to EBADMODE, regardless of the file type.

If the mode string that is passed to fopen() is not valid, fopen() will set errno to EBADMODE, regardless of the file type.

If the mode string passed to fopen() is correct, but is invalid to that specific type of file, fopen() will set errno to ENOTOPEN, EIOERROR, or EIORECERR, regardless of the file type.

Example that uses fopen()

This example attempts to open a file for reading.

```
#include <stdio.h>
#define MAX LEN 60
int main(void)
  FILE *stream;
  fpos t pos;
  char line1[MAX LEN];
  char line2[MAX_LEN];
  char *result;
  char ch;
  int num;
  /* The following call opens a text file for reading.
   if ((stream = fopen("mylib/myfile", "r")) == NULL)
      printf("Could not open data file\n");
  else if ((result = fgets(line1,MAX_LEN,stream)) != NULL)
            printf("The string read from myfile: %s\n", result);
            fclose(stream);
  /* The following call opens a fixed record length file */
  /* for reading and writing.
  if ((stream = fopen("mylib/myfile2", "rb+, lrecl=80, \
                 blksize=240, recfm=f")) == NULL)
         printf("Could not open data file\n");
  else {
         fgetpos(stream, Point-of-Sale);
         if (!fread(line2,sizeof(line2),1,stream))
         perror("fread error");
else printf("1st record read from myfile2: %s\n", line2);
         fsetpos(stream, Point-of-Sale);
                                            /* Reset pointer to start of file */
         fputs(result, stream); /* The line read from myfile is */
                                    /* written to myfile2.
         fclose(stream);
}
```

Related Information

- "fclose() Close Stream" on page 75
- "fflush() Write Buffer to File" on page 80
- "fread() Read Items" on page 108
- "freopen() Redirect Open Files" on page 111
- "fseek() fseeko() Reposition File Position" on page 114
- "fsetpos() Set File Position" on page 117
- "fwrite() Write Items" on page 127
- "rewind() Adjust Current File Position" on page 245
- "<stdio.h>" on page 15
- "wfopen() —Open Files" on page 442
- open() API in the Information Center API topic.

fprintf() — Write Formatted Data to a Stream

Format

```
#include <stdio.h>
int fprintf(FILE *stream, const char *format-string, argument-list);
```

Language Level: ANSI

Thread Safe: YES

Description

1

The fprintf() function formats and writes a series of characters and values to the output stream. The fprintf() function converts each entry in argument-list, if any, and writes to the stream according to the corresponding format specification in the format-string.

The format-string has the same form and function as the format-string argument for the printf() function.

Return Value

The fprintf() function returns the number of bytes that are printed or a negative value if an output error occurs.

Example that uses fprintf()

This example sends a line of asterisks for each integer in the array count to the file myfile. The number of asterisks that are printed on each line corresponds to an integer in the array.

```
#include <stdio.h>
int count [10] = \{1, 5, 8, 3, 0, 3, 5, 6, 8, 10\};
int main(void)
  int i,j;
  FILE *stream;
  stream = fopen("mylib/myfile", "w");
                 /* Open the stream for writing */
  for (i=0; i < sizeof(count) / sizeof(count[0]); i++)</pre>
     for (j = 0; j < count[i]; j++)
        fprintf(stream, "*");
                /* Print asterisk
                                                */
        fprintf(stream, "\n");
                 /* Move to the next line
                                                */
  fclose (stream);
/******* Output should be similar to: ********
****
******
***
****
*****
*****
*/
```

Related Information

• "fscanf() — Read Formatted Data" on page 113

- "fwprintf() Format Data as Wide Characters and Write to a Stream" on
- "printf() Print Formatted Characters" on page 200
- "sprintf() Print Formatted Data to Buffer" on page 317
- "vfprintf() Print Argument Data to Stream" on page 386
- "vprintf() Print Argument Data" on page 389
- "vsprintf() Print Argument Data to Buffer" on page 391
- "<stdio.h>" on page 15

fputc() — Write Character

Format

#include <stdio.h> int fputc(int c, FILE *stream);

Language Level: ANSI

Thread Safe: YES

Description

The fputc() function converts *c* to an unsigned char and then writes *c* to the output stream at the current position and advances the file position appropriately. If the stream is opened with one of the append modes, the character is appended to the end of the stream.

The fputc() function is identical to putc(); it always is defined as a function call; it is never replaced by a macro.

Return Value

The fputc() function returns the character that is written. A return value of EOF indicates an error.

The value of errno may be set to:

Value Meaning

ENOTWRITE

The file is not open for write operations.

EPUTANDGET

A write operation that was not permitted occurred after a read operation.

ERECIO

The file is open for record I/O.

ESTDERR

stderr cannot be opened.

ESTDOUT

stdout cannot be opened.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

The fputc() function is not supported for files that are opened with type=record.

Example that uses fputc()

This example writes the contents of buffer to a file that is called *myfile*.

Note: Because the output occurs as a side effect within the second expression of the for statement, the statement body is null.

```
#include <stdio.h>
#define NUM_ALPHA 26
int main(void)
 FILE * stream;
 int i;
 int ch;
 char buffer[NUM ALPHA + 1] = "abcdefghijklmnopqrstuvwxyz";
 if (( stream = fopen("mylib/myfile", "w"))!= NULL )
   /* Put buffer into file */
   for ( i = 0; ( i < sizeof(buffer) ) &&
           ((ch = fputc( buffer[i], stream)) != EOF ); ++i );
    fclose( stream );
 else
   perror( "Error opening myfile" );
```

Related Information

- "fgetc() Read a Character" on page 82
- "putc() putchar() Write a Character" on page 210
- "<stdio.h>" on page 15

_fputchar - Write Character

Format

```
#include <stdio.h>
int _fputchar(int c);
```

Note: The _fputchar function is supported only for C++, not for C.

Language Level: Extension

Description

_fputchar writes the single character c to the stdout stream at the current position. It is equivalent to the following fputc call:

```
fputc(c, stdout);
```

For portability, use the ANSI/ISO fputc function instead of _fputchar.

Return Value

_fputchar returns the character written. A return value of EOF indicates that a write error has occurred. Use ferror and feof to tell whether this is an error condition or the end of the file.

Example that uses _fputchar()

This example writes the contents of buffer to stdout:

```
#include <stdio.h>
int main(void)
{
    char buffer[80];
    int i,ch = 1;
    for (i = 0; i < 80; i++)
        buffer[i] = 'c';
    for (i = 0; (i < 80) && (ch != EOF); i++)
        ch = _fputchar(buffer[i]);
    printf("\n");
    return 0;
}</pre>
```

The output should be similar to:

Related Information:

- "getc() getchar() Read a Character" on page 133
- "fputc() Write Character" on page 101
- "putc() putchar() Write a Character" on page 210
- "<stdio.h>" on page 15

fputs() — Write String

Format

```
#include <stdio.h>
int fputs(const char *string, FILE *stream);
```

Language Level: ANSI

Thread Safe: YES

Description

The fputs () function copies *string* to the output *stream* at the current position. It does not copy the null character (\0) at the end of the string.

Return Value

The fputs() function returns EOF if an error occurs; otherwise, it returns a non-negative value.

The value of errno may be set to:

Value Meaning

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

EPUTANDGET

A write operation that was not permitted occurred after a read operation.

The fputs() function is not supported for files that are opened with type=record.

Example that uses fputs()

This example writes a string to a stream.

```
#include <stdio.h>
#define NUM ALPHA 26
int main(void)
 FILE * stream;
 int num;
 /* Do not forget that the '\0' char occupies one character */
 static char buffer[NUM ALPHA + 1] = "abcdefghijklmnopqrstuvwxyz";
 if ((stream = fopen("mylib/myfile", "w")) != NULL )
     /* Put buffer into file */
    if ( (num = fputs( buffer, stream )) != EOF )
      /* Note that fputs() does not copy the \0 character */
      printf( "Total number of characters written to file = %i\n", num );
      fclose( stream );
    else /* fputs failed */
      perror( "fputs failed" );
 else
    perror( "Error opening myfile" );
```

Related Information

- "fgets() Read a String" on page 85
- "fputws() Write Wide-Character String" on page 106
- "gets() Read a Line" on page 137
- "puts() Write a String" on page 212
- "<stdio.h>" on page 15

fputwc() — Write Wide Character

Format

```
#include <wchar.h>
#include <stdio.h>
wint t fputwc(wint t wc, FILE *stream);
```

Language Level: ANSI

Thread Safe: YES

Description

The fputwc() function converts the wide character wc to a multibyte character and writes it to the output stream pointed to by stream at the current position. It also

advances the file position indicator appropriately. If the file cannot support positioning requests, or if the stream was opened with append mode, the character is appended to the stream.

Using non-wide-character functions with the fputwc() function on the same stream will result in undefined behavior. After calling the fputwc() function, delete the buffer or reposition the stream pointer before calling a read function for the stream. After reading from the stream, delete the buffer or reposition the stream pointer before calling the fputwc() function, unless EOF has been reached.

Note: The behavior of the fputwc() function is affected by the LC_CTYPE category of the current locale. If you change the category between subsequent operations on the same stream, undefined results can occur. This function is available when SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) are specified on the compilation command.

Return Value

The fputwc() function returns the wide character that is written. If a write error occurs, the error indicator for the stream is set, and the fputwc() function returns WEOF. If an encoding error occurs during conversion from wide character to a multibyte character, fputwc() sets errno to EILSEQ and returns WEOF.

Example that uses fputwc()

This example opens a file and uses the fputwc() function to write wide characters to the file.

```
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <errno.h>
int main(void)
  FILE
       *stream;
  wchar_t *wcs = L"A character string.";
  if (NULL == (stream = fopen("fputwc.out", "w")))
     printf("Unable to open: \"fputwc.out\".\n");
     exit(1);
  for (i = 0; wcs[i] != L' \setminus 0'; i++) {
     errno = 0;
     if (WEOF == fputwc(wcs[i], stream)) {
        printf("Unable to fputwc() the wide character.\n" "wcs[%d] = 0x\%.41x\n", i, wcs[i]);
        if (EILSEQ == errno)
          printf("An invalid wide character was encountered.\n");
        exit(1);
  fclose(stream);
  return 0;
  /*********************
     The output file fputwc.out should contain:
     A character string.
```

Related Information

- "fgetwc() Read Wide Character from Stream" on page 86
- "fputc() Write Character" on page 101
- "fputwc() Write Wide Character" on page 104
- "putc() putchar() Write a Character" on page 210
- "putwchar() Write Wide Character to stdout" on page 214
- "putwc() Write Wide Character" on page 213
- "<stdio.h>" on page 15
- "<wchar.h>" on page 18

fputws() — Write Wide-Character String

```
Format
```

```
#include <wchar.h>
#include <stdio.h>
int fputws(const wchar t *wcs, FILE *stream);
```

Language Level: XPG4

Thread Safe: YES.

Description

The fputws() function converts the wide-character string wcs to a multibyte-character string and writes it to stream as a multibyte-character string. It does not write the ending null wide character.

Using non-wide-character functions with the fputws () function on the same stream will result in undefined behavior. After calling the fputws () function, flush the buffer or reposition the stream pointer before calling a read function for the stream. After a read operation, flush the buffer or reposition the stream pointer before calling the fputws () function, unless EOF has been reached.

Note: The behavior of the fputws() function is affected by the LC_CTYPE category of the current locale. If you change the category between subsequent operations on the same stream, undefined results can occur. This function is available when SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) are specified on the compilation command.

Return Value

The fputws () function returns a non-negative value if successful. If a write error occurs, the error indicator for the stream is set, and the fputws() function returns -1. If an encoding error occurs in converting the wide characters to multibyte characters, the fputws () function sets errno to EILSEQ and returns -1.

Example that uses fputws()

This example opens a file and writes a wide-character string to the file using the fgetws() function.

```
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <errno.h>
int main(void)
  FILE
       *stream;
  wchar t *wcs = L"This test string should not return -1";
  if (NULL == (stream = fopen("fputws.out", "w"))) {
     printf("Unable to open: \"fputws.out\".\n");
     exit(1);
  errno = 0;
  if (EOF == fputws(wcs, stream)) {
     printf("Unable to complete fputws() function.\n");
     if (EILSEQ == errno)
        printf("An invalid wide character was encountered.\n");
     exit(1);
  fclose(stream);
  return 0;
  /*********************
     The output file fputws.out should contain:
     This test string should not return -1
  **************************************
```

Related Information

- "fgetws() Read Wide-Character String from Stream" on page 88
- "fputs() Write String" on page 103
- "fputwc() Write Wide Character" on page 104
- "puts() Write a String" on page 212
- "<stdio.h>" on page 15
- "<wchar.h>" on page 18

fread() — Read Items

Format

#include <stdio.h> size t fread(void *buffer, size t size, size t count, FILE *stream);

Language Level: ANSI

Thread Safe: YES

Description

The fread() function reads up to *count* items of *size* length from the input *stream* and stores them in the given buffer. The position in the file increases by the number of bytes read.

Return Value

The fread() function returns the number of full items successfully read, which can be less than *count* if an error occurs, or if the end-of-file is met before reaching count. If size or count is 0, the fread() function returns zero, and the contents of the array and the state of the stream remain unchanged.

The value of errno may be set to:

Value Meaning

EGETANDPUT

A read operation that was not permitted occurred after a write operation.

ENOREC

Record is not found.

ENOTREAD

The file is not open for read operations.

ERECIO

The file is open for record I/O.

ESTDIN

stdin cannot be opened.

ETRUNC

Truncation occurred on the operation.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

Use the ferror() and feof() functions to distinguish between a read error and an end-of-file.

When using fread() for record input, set *size* to 1 and *count* to the maximum expected length of the record, to obtain the number of bytes. If you do not know the record length, you should set *size* to 1 and *count* to a large value. You can read only one record at a time when using record I/O.

Example that uses fread()

This example attempts to read NUM_ALPHA characters from the file *myfile*. If there are any errors with either fread() or fopen(), a message is printed.

```
#include <stdio.h>
#define NUM ALPHA 26
int main(void)
 FILE * stream;
                 /* number of characters read from stream */
 int num:
  /* Do not forget that the '\0' char occupies one character too! */
 char buffer[NUM ALPHA + 1];
  if (( stream = fopen("mylib/myfile", "r"))!= NULL )
   memset(buffer, 0, sizeof(buffer));
   num = fread( buffer, sizeof( char ), NUM_ALPHA, stream );
    if ( num ) { /* fread success */
      printf( "Number of characters has been read = %i\n", num );
      printf( "buffer = %s\n", buffer );
      fclose( stream );
    else { /* fread failed */
      if ( ferror(stream) )
                              /* possibility 1 */
      perror( "Error reading myfile" );
else if ( feof(stream)) /* possibility 2 */
        perror( "EOF found" );
    }
   perror( "Error opening myfile" );
```

Related Information

- "feof() Test End-of-File Indicator" on page 79
- "ferror() Test for Read/Write Errors" on page 79
- "fopen() Open Files" on page 92
- "fwrite() Write Items" on page 127
- "<stdio.h>" on page 15

free() — Release Storage Blocks

Format

```
#include <stdlib.h>
void free(void *ptr);
```

Language Level: ANSI

Thread Safe: YES

Description

1

The free() function frees a block of storage. The ptr argument points to a block that is previously reserved with a call to the calloc(), malloc(), realloc(), _C_TS_calloc(), _C_TS_malloc(), _C_TS_realloc(), or _C_TS_malloc64() functions. The number of bytes freed is the number of bytes specified when you reserved (or reallocated, in the case of the realloc() function) the block of storage. If ptr is NULL, free() simply returns.

Note: Attempting to free a block of storage not allocated with calloc(), malloc(), or realloc() (or previously freed storage) can affect the subsequent reserving of storage and lead to undefined results. Storage that is allocated with the ILE bindable API CEEGTST can be freed with free().

To use **Teraspace** storage instead of heap storage without changing the C source code, specify the TERASPACE(*YES *TSIFC) parameter on the CRTCMOD compiler command. This maps the free() library function to C TS free(), its Teraspace storage counterpart.

Return Value

There is no return value.

Example that uses free()

This example uses the calloc() function to allocate storage for x array elements, and then calls the free() function to free them.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
               /* start of the array
/* index variable
 long * array;
 long * index;
                /* index variable
 int i;
 int num;
                /* number of entries of the array */
 printf( "Enter the size of the array\n" );
 scanf( "%i", &num );
 /* allocate num entries */
 if ( (index = array = calloc( num, sizeof( long ))) != NULL )
   for ( i = 0; i < num; ++i ) /* put values in array
      *index++ = i;
                                      /* using pointer notation */
   free( array );
                                       /* deallocates array
 else
 { /* Out of storage */
   perror( "Error: out of storage" );
   abort();
```

Related Information

- "calloc() Reserve and Initialize Storage" on page 56
- "malloc() Reserve Storage Block" on page 170

- "realloc() Change Reserved Storage Block Size" on page 234
- "<stdlib.h>" on page 16

freopen() — Redirect Open Files

Format

```
#include <stdio.h>
FILE *freopen(const char *filename, const char *mode, FILE *stream);
```

Language Level: ANSI

Description

The freopen() function closes the file that is currently associated with *stream* and reassigns *stream* to the file that is specified by *filename*. The freopen() function opens the new file associated with *stream* with the given *mode*, which is a character string specifying the type of access requested for the file. You can also use the freopen() function to redirect the standard stream files stdin, stdout, and stderr to files that you specify.

For database files, if *filename* is an empty string, the freopen() function closes and reopens the stream to the new open mode, rather than reassigning it to a new file or device. You can use the freopen() function with no file name specified to change the mode of a standard stream from text to binary without redirecting the stream, for example:

```
fp = freopen("", "rb", stdin);
```

You can use the same method to change the mode from binary back to text.

You cannot use the freopen() function with *filename* as an empty string in modules created with SYSIFCOPT(*IFSIO).

You can use the same method to change the mode from binary back to text.

Return Value

The freopen() function returns a pointer to the newly opened stream. If an error occurs, the freopen() function closes the original file and returns a NULL pointer value.

The value of errno may be set to:

Value Meaning

EBADF

1

The file pointer or descriptor is not valid.

EBADMODE

The file mode that is specified is not valid.

EBADNAME

The file name that is specified is not valid.

ENOENT

No file or library.

ENOTOPEN

The file is not open.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

Example that uses freopen()

This example closes the *stream1* data stream and reassigns its stream pointer. stream1 and stream2 will have the same value, but they will not necessarily have the same value as stream.

```
#include <stdio.h>
#define MAX LEN 100
int main(void)
  FILE *stream, *stream1, *stream2;
  char line[MAX LEN], *result;
  int i;
  stream = fopen("mylib/myfile","r");
   if ((result = fgets(line, MAX LEN, stream)) != NULL)
       printf("The string is %s\n", result);
   /* Change all spaces in the line to '*'. */
   for (i=0; i<=sizeof(line); i++)</pre>
     if (line[i] == ' ')
        line[i] = '*';
  stream1 = stream;
  stream2 = freopen("", "w+", stream1);
   fputs( line, stream2 );
   fclose( stream2);
```

Related Information

- "fclose() Close Stream" on page 75
- "fopen() Open Files" on page 92
- "<stdio.h>" on page 15

frexp() — Separate Floating-Point Value

Format

```
#include <math.h>
double frexp(double x, int *expptr);
```

Language Level: ANSI

Description

The frexp() function breaks down the floating-point value x into a term m for the mantissa and another term n for the exponent. It is done such that $x=m*2^n$, and the absolute value of m is greater than or equal to 0.5 and less than 1.0 or equal to 0. The frexp() function stores the integer exponent n at the location to which expptr points.

Return Value

The frexp() function returns the mantissa term m. If x is 0, frexp() returns 0 for both the mantissa and exponent. The mantissa has the same sign as the argument *x*. The result of the frexp() function cannot have a range error.

Example that uses frexp()

This example separates the floating-point value of x, 16.4, into its mantissa 0.5125, and its exponent 5. It stores the mantissa in y and the exponent in n.

```
#include <math.h>
#include <stdio.h>
int main(void)
{
  double x, m;
  int n;
  x = 16.4;
  m = frexp(x, n);
  printf("The mantissa is lf and the exponent is d\n", m, n);
/******* Output should be similar to: ********
The mantissa is 0.512500 and the exponent is 5
```

Related Information

- "ldexp() Multiply by a Power of Two" on page 156
- "modf() Separate Floating-Point Value" on page 195
- "<math.h>" on page 8

fscanf() — Read Formatted Data

Format

```
#include <stdio.h>
int fscanf (FILE *stream, const char *format-string, argument-list);
```

Language Level: ANSI

Thread Safe: YES

Description

The fscanf() function reads data from the current position of the specified *stream* into the locations that are given by the entries in argument-list, if any. Each entry in argument-list must be a pointer to a variable with a type that corresponds to a type specifier in *format-string*.

The format-string controls the interpretation of the input fields and has the same form and function as the *format-string* argument for the scanf() function.

Return Value

The fscanf() function returns the number of fields that it successfully converted and assigned. The return value does not include fields that the fscanf() function read but did not assign.

The return value is EOF if an input failure occurs before any conversion, or the number of input items assigned if successful.

Example that uses fscanf()

This example opens the file *myfile* for reading and then scans this file for a string, a long integer value, a character, and a floating-point value.

```
#include <stdio.h>
#define MAX LEN 80
int main(void)
  FILE *stream;
  long 1;
  float fp;
  char s[MAX LEN + 1];
  char c;
  stream = fopen("mylib/myfile", "r");
  /* Put in various data. */
  fscanf(stream, "%s", &s [0]);
  fscanf(stream, "%1d", &1);
fscanf(stream, "%c", &c);
fscanf(stream, "%f", &fp);
  printf("string = %s\n", s);
  printf("long double = %ld\n", 1);
  printf("char = %c\n", c);
  printf("float = %f\n", fp);
****** abcdefghijklmnopqrstuvwxyz 343.2 ********
****** expected output is: **********
string = abcdefghijklmnopqrstuvwxyz
long double = 343
char = .
float = 2.000000
*/
```

Related Information

- "fprintf() Write Formatted Data to a Stream" on page 99
- "fwscanf() Read Data from Stream Using Wide Character" on page 128
- "scanf() Read Data" on page 299
- "sscanf() Read Data" on page 322
- "<stdio.h>" on page 15
- "swscanf() Read Wide Character Data" on page 369
- "wscanf() Read Data Using Wide-Character Format String" on page 448

fseek() — fseeko() — Reposition File Position

```
Format
```

```
#include <stdio.h>
int fseek(FILE *stream, long int offset, int origin);
int fseeko(FILE *stream, off_t offset, int origin);
```

Language Level: ANSI I Description The fseek() and fseeko() functions change the current file position that is associated with stream to a new location within the file. The next operation on stream takes place at the new location. On a stream open for update, the next operation can be either a reading or a writing operation. The fseeko() function is identical to fseek() except that the offset argument is of type off_t_. The *origin* must be one of the following constants that are defined in <stdio.h>: Origin Definition SEEK_SET Beginning of file SEEK CUR Current position of file pointer **SEEK END** End of file For a binary stream, you can also change the position beyond the end of the file. An attempt to position before the beginning of the file causes an error. If successful, the fseek() or fseeko() function clears the end-of-file indicator, even ı when *origin* is SEEK_END, and undoes the effect of any preceding the ungetc() function on the same stream. Note: For streams opened in text mode, the fseek() and fseeko() functions have limited use because some system translations (such as those between carriage-return-line-feed and new line) can produce unexpected results. The only fseek() and fseeko() operations that can be relied upon to work on streams opened in text mode are seeking with an offset of zero relative to any of the origin values, or seeking from the beginning of the file with an offset value returned from a call to the ftell()or ftello() functions. Calls to the ftell() and ftello() functions are subject to their restrictions. Return Value The fseek() or fseeko function returns 0 if it successfully moves the pointer. A nonzero return value indicates an error. On devices that cannot seek, such as terminals and printers, the return value is nonzero. The value of errno may be set to: Value Meaning **EBADF** The file pointer or descriptor is invalid. **EBADSEEK** ı Bad offset for a seek operation. **ENODEV**

Operation was attempted on a wrong device.

ENOTOPEN

The file is not open.

ERECIO

The file is open for record I/O.

ESTDERR

stderr cannot be opened.

ESTDIN

stdin cannot be opened.

ESTDOUT

stdout cannot be opened.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

The fseek() and fseeko() functions are not supported for files that are opened with type=record.

Example that uses fseek()

This example opens a file myfile for reading. After performing input operations, fseek() moves the file pointer to the beginning of the file.

```
#include <stdio.h>
#define MAX LEN 10
int main(void)
  FILE *stream;
  char buffer[MAX_LEN + 1];
  int result;
  int i;
  char ch;
  stream = fopen("mylib/myfile", "r");
   for (i = 0; (i < (sizeof(buffer)-1) &&
       ((ch = fgetc(stream)) != EOF) && (ch != '\n')); i++)
  result = fseek(stream, 0L, SEEK_SET); /* moves the pointer to the */
                                          /* beginning of the file
   if (result == 0)
     printf("Pointer successfully moved to the beginning of the file.\n");
     printf("Failed moving pointer to the beginning of the file.\n");
```

Related Information

- "ftell() ftello() Get Current Position" on page 118
- "fgetpos() Get File Position" on page 84
- "fsetpos() Set File Position" on page 117
- "rewind() Adjust Current File Position" on page 245
- "ungetc() Push Character onto Input Stream" on page 381
- "<stdio.h>" on page 15
- "fseek() fseeko() Reposition File Position" on page 114

fsetpos() — Set File Position

Format

```
#include <stdio.h>
int fsetpos(FILE *stream, const fpos_t *pos);
```

Language Level: ANSI

Description

The fsetpos() function moves any file position that is associated with stream to a new location within the file according to the value pointed to by pos. The value of pos was obtained by a previous call to the fgetpos() library function.

If successful, fsetpos() clears the end-of-file indicator, and undoes the effect of any previous ungetc() function on the same stream.

After the fsetpos() call, the next operation on a stream in update mode may be input or output.

Return Value

If fsetpos() successfully changes the current position of the file, it returns 0. A nonzero return value indicates an error.

The value of errno may be set to:

Value Meaning

EBADF

I

The file pointer or descriptor is invalid..

EBADPOS

The position that is specified is not valid.

EINVAL

The value specified for the argument is not correct. You may receive this errno when you compile your program with *IFSIO, and you are working with a file in the QSYS file system. For example,

"/qsys.lib/qtemp.lib/myfile.file/mymem.mbr"

ENODEV

Operation was attempted on a wrong device.

ENOPOS

No record at the specified position.

ERECIO

The file is open for record I/O.

ESTDERR

stderr cannot be opened.

ESTDIN

stdin cannot be opened.

ESTDOUT

stdout cannot be opened.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

The fsetpos() function cannot be used for files that are opened with type=record. Also, the fsetpos() function can only support setting the position to the beginning of the file if:

- your program is compiled with *IFSIO, and
- you are working on a file in the QSYS file system.

Example that uses fsetpos()

This example opens a file mylib/myfile for reading. After performing input operations, fsetpos() moves the file pointer to the beginning of the file and rereads the first byte.

```
#include <stdio.h>
FILE *stream;
int main(void)
   int retcode;
  fpos t pos;
  char ptr[20]; /* existing file 'mylib/myfile' has 20 byte records */
  int i;
  /* Open file, get position of file pointer, and read first record */
  stream = fopen("mylib/myfile", "rb");
   fgetpos(stream, Point-of-Sale);
   if (!fread(ptr,sizeof(ptr),1,stream))
       perror("fread error");
  else printf("1st record: %s\n", ptr);
   /* Perform another read operation on the second record
   /* - the value of 'pos' changes
   if (!fread(ptr,sizeof(ptr),1,stream))
  perror("fread error");
else printf("2nd record: %s\n", ptr);
   /* Re-set pointer to start of file and re-read first record */
   fsetpos(stream, Point-of-Sale);
   if (!fread(ptr,sizeof(ptr),1,stream))
       perror("fread error");
   else printf("1st record again: %s\n", ptr);
   fclose(stream);
```

Related Information

- "fgetpos() Get File Position" on page 84
- "fseek() fseeko() Reposition File Position" on page 114
- "ftell() ftello() Get Current Position"
- "rewind() Adjust Current File Position" on page 245
- "<stdio.h>" on page 15

ftell() — ftello() — Get Current Position

Format

#include <stdio.h> long int ftell(FILE *stream); off t ftello(FILE *stream); Language Level: ANSI Description The ftell() and ftello() functions find the current position of the file associated with stream. For a fixed-length binary file, the value that is returned is an offset ı relative to the beginning of the stream. For file in the QSYS library system, the ftell() and ftello() functions return a relative value for fixed-format binary files and an encoded value for other file types. This encoded value must be used in calls to the fseek() and fseeko() functions to positions other than the beginning of the file. The ftell() and ftello() functions will fail if the file is not a database file or Integrated File System file. Note: The ftello() function is available only when SYSIFCOPT(*IFSIO) is specified on the compilation command. Return Value The ftell() and ftello() functions return the current file position. On error, ftell() and ftello() return -1, cast to long and off t respectively, and set errno to a nonzero value. The value of errno may be set to: Value Meaning **ENODEV** Operation was attempted on a wrong device. **ENOTOPEN** The file is not open. **ENUMMBRS** The file is open for multi-member processing. ı **ENUMRECS** Too many records. **ERECIO** The file is open for record I/O. **ESTDERR** stderr cannot be opened. **ESTDIN** stdin cannot be opened. **ESTDOUT** stdout cannot be opened. **EIOERROR** A non-recoverable I/O error occurred. **EIORECERR**

A recoverable I/O error occurred.

The ftell() and ftello() functions are not supported for files that are opened with type=record.

Example that uses ftell()

This example opens the file mylib/myfile for reading. It reads enough characters to fill half of the buffer and prints out the position in the stream and the buffer.

```
#include <stdio.h>
#define NUM ALPHA 26
#define NUM CHAR
int main(void)
 FILE * stream;
 int i;
 char ch;
 char buffer[NUM ALPHA];
 long position;
  if (( stream = fopen("mylib/myfile", "r")) != NULL )
    /* read into buffer */
    for ( i = 0; ( i < NUM_ALPHA/2 ) &&
                                                    ((buffer[i] = fgetc(stream)) != EOF ); ++i )
        if (i==NUM_CHAR-1) ^{-}/* We want to be able to position the ^{*}/
                                     /* file pointer to the character in
   */
                                     /* position NUM_CHAR
   */
           position = ftell(stream);
   buffer[i] = '\0';
```

Related Information

- "fseek() fseeko() Reposition File Position" on page 114
- "fgetpos() Get File Position" on page 84
- "fopen() Open Files" on page 92
- "fsetpos() Set File Position" on page 117
- "<stdio.h>" on page 15
- "ftell() ftello() Get Current Position" on page 118

fwide() — Determine Stream Orientation

Format

```
#include <stdio.h>
#include <wchar.h>
int fwide(FILE *stream, int mode);
```

Language Level: ANSI

Description

The fwide() function determines the orientation of the stream pointed to by stream. If *mode* is greater than 0, the fwide() function first attempts to make the stream wide oriented. If mode is less than 0, the fwide() function first attempts to make the stream byte oriented.

Note: If the orientation of the stream has already been determined, the fwide() function does not change it.

Otherwise, mode is 0, and the fwide() function does not alter the orientation of the stream.

Note: This function is available when SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) are specified on the compilation command.

Return Value

If, after the call, the stream has wide orientation, the fwide() function returns a value greater than 0. If the stream has byte orientation, it returns a value less than 0. If the stream has no orientation, it returns 0.

Example that uses fwide()

```
#include <stdio.h>
#include <math.h>
#include <wchar.h>
void check orientation(FILE *stream)
  int rc;
                            /* check the orientation */
  rc = fwide(stream,0);
  if (rc<0)
     printf("Stream has byte orientation.\n");
  } else if (rc>0) {
   printf("Stream has wide orientation.\n");
   } else {
     printf("Stream has no orientation.\n");
  return;
int main(void)
  FILE *stream;
  /* Demonstrate that fwide can be used to set the orientation,
     but cannot change it once it has been set.
  stream = fopen("test.dat", "w");
  printf("After opening the file: ");
  check_orientation(stream);
  fwide(stream, -1);
                           /* Make the stream byte oriented */
  printf("After fwide(stream, -1): ");
  check orientation(stream);
                           /* Try to make the stream wide oriented */
  fwide(stream, 1);
  printf("After fwide(stream, 1): ");
  check orientation(stream);
  fclose(stream);
  printf("Close the stream\n");
  /* Check that a wide character output operation sets the orientation
     as expected. */
  stream = fopen("test.dat","w");
  printf("After opening the file: ");
  check orientation(stream);
  fwprintf(stream, L"pi = %.5f\n", 4* atan(1.0));
  printf("After fwprintf(): ");
  check orientation(stream);
  fclose(stream);
  return 0;
  The output should be similar to :
     After opening the file: Stream has no orientation.
     After fwide(stream, -1): Stream has byte orientation. After fwide(stream, 1): Stream has byte orientation.
     Close the stream
     After opening the file: Stream has no orientation.
     After fwprintf(): Stream has wide orientation.
   }
```

Related Information

- "fgetwc() Read Wide Character from Stream" on page 86
- "fgetws() Read Wide-Character String from Stream" on page 88
- "fputwc() Write Wide Character" on page 104
- "fputws() Write Wide-Character String" on page 106
- "<stdio.h>" on page 15
- "<wchar.h>" on page 18

fwprintf() — Format Data as Wide Characters and Write to a Stream

Format

```
#include <stdio.h>
#include <wchar.h>
int fwprintf(FILE *stream, const wchar t *format, argument-list);
```

Language Level: ANSI

Thread Safe: YES

Description

ı

The fwprintf() function writes output to the stream pointed to by stream, under control of the wide string pointed to by format. The format string specifies how subsequent arguments are converted for output.

The fwprintf() function converts each entry in argument-list according to the corresponding wide-character format specifier in format.

If insufficient arguments exist for the format, the behavior is undefined. If the format is exhausted while arguments remain, the fwprintf() function evaluates, as usual, the excess arguments, but otherwise ignores them. The fwprintf() function returns when it encounters the end of the format string.

The format comprises zero or more directives: ordinary wide characters (not %) and conversion specifications. Conversion specifications are processed as if they were replaced in the format string by wide-character strings. The wide-character strings are the result of fetching zero or more subsequent arguments and then converting them, if applicable, according to the corresponding conversion specifier. The fwprintf() function then writes the expanded wide-character format string to the output stream.

The format for the fwprintf() function has the same form and function as the format string for printf(), with the following exceptions:

- %c (without an l prefix) converts an integer argument to wchar_t, as if by calling the btowc() function.
- %s (without an 1 prefix) converts an array of multibyte characters to an array of wchar_t, as if by calling the mbrtowc() function. The array is written up to, but not including, the terminating null character, unless the precision specifies a shorter output.
- %ls and %S write an array of wchar_t. The array is written up to, but not including, the ending null character, unless the precision specifies a shorter output.

If a conversion specification is invalid, the behavior is undefined.

If any argument is, or points to, an union or an aggregate (except for an array of char type using %s conversion, an array of wchar_t type using %ls conversion, or a pointer using %p conversion), the behavior is undefined.

In no case does a nonexistent, or small field width, cause truncation of a field; if the conversion result is wider than the field width, the field is expanded to contain the conversion result.

Note: This function is available when SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) are specified on the compilation command.

Return Value

The fwprintf() function returns the number of wide characters transmitted. If an output error occurred, it returns a negative value.

Example that uses fwprintf()

Note: When writing wide characters, the file should be opened in binary mode, or the file must be opened with the o_ccsid or codepage parameters. This ensures that no conversions occur on your wide characters. Since there are no valid double-byte job CCSIDS on the iSeries, any conversion would be invalid.

```
#include <stdio.h>
#include <wchar.h>
#include <locale.h>
int count [10] = \{1, 5, 8, 3, 0, 3, 5, 6, 8, 10\};
int main(void)
  int i,j;
  FILE *stream;
  /* Open the stream for writing */
  if (NULL == (stream = fopen("/QSYS.LIB/LIB.LIB/WCHAR.FILE/WCHAR.MBR","wb")))
     perror("fopen error");
  for (i=0; i < sizeof(count) / sizeof(count[0]); i++)</pre>
  {
      for (j = 0; j < count[i]; j++)
         fwprintf(stream, L"*");
                  /* Print asterisk
                                                 */
         fwprintf(stream, L"\n");
                  /* Move to the next line
                                                 */
  }
  fclose (stream);
}
/*
The member WCHAR of file WCHAR will contain:
****
******
***
***
                                                        Chapter 2. Library Functions 125
```

****** *****

Unicode example that uses fwprintf()

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
/* This program is compile LOCALETYPE(*LOCALEUCS2) and
/* SYSIFCOPT(*IFSIO)
int main(void)
   FILE *stream;
   wchar t wc = 0x0058; /* UNICODE X */
   char c1 = 'c';
   char *s1 = "123";
   wchar t ws[4];
   setlocale(LC_ALL,
    "/QSYS.LIB/EN_US.LOCALE"); /* a CCSID 37 locale */
                     /* UNICODE A */
   ws[0] = 0x0041;
   ws[1] = (wchar_t)0x0042; /* UNICODE B
                                   /* UNICODE C */
   ws[2] = (wchar^{-}t)0x0043;
   ws[3] = (wchar_t)0x0000;
   stream = fopen("myfile.dat", "wb+");
   /* Ic and Is take wide char as input and just copies then */
   /* to the file. So the file would look like this
                                                            */
   /* after the below fwprintf statement:
                                                            */
   /* 0058002000200020004100420043
                                                            */
   /* 0020 is UNICODE blank
                                                            */
   fwprintf(stream, L"%lc %ls",wc,ws);
   /* c and s take multibyte as input and produce UNICODE
   /* In this case c1 and s1 are CCSID 37 characters based */
   /* on the setlocale above. So the characters are
                                                            */
   /* converted from CCSID 37 to UNICODE and will look
                                                            */
   /* like this in hex after the following fwprintf
                                                            */
   /* statment: 0063002000200020003100320033
                                                            */
   /* 0063 is a UNICODE c 0031 is a UNICODE 1 and so on
                                                            */
   fwprintf(stream, L"%c %s",c1,s1);
   /* Now lets try width and precision. 6ls means write */
   /* 6 wide characters so we will pad with 3 UNICODE
   /* blanks and %.2s means write no more then 2 wide
                                                          */
   /* characters. So we get an output that looks like
   /* this: 00200020002000410042004300310032
   fwprintf(stream, L"%61s%.2s",ws,s1);
```

Related Information

- "fprintf() Write Formatted Data to a Stream" on page 99
- "printf() Print Formatted Characters" on page 200
- "vfprintf() Print Argument Data to Stream" on page 386
- "vprintf() Print Argument Data" on page 389
- "btowc() Convert Single Byte to Wide Character" on page 54
- "mbrtowc() Convert a Multibyte Character to a Wide Character (Restartable)" on page 176
- "vfwprintf() Format Argument Data as Wide Characters and Write to a Stream" on page 387
- "vswprintf() Format and Write Wide Characters to Buffer" on page 393
- "wprintf() Format Data as Wide Characters and Print" on page 447

- "<stdarg.h>" on page 14
- "<wchar.h>" on page 18

fwrite() — Write Items

I

Format

```
#include <stdio.h>
size t fwrite(const void *buffer, size t size, size t count,
                 FILE *stream);
```

Language Level: ANSI

Thread Safe: YES

Description

The fwrite() function writes up to count items, each of size bytes in length, from buffer to the output stream.

Return Value

The fwrite() function returns the number of full items successfully written, which can be fewer than count if an error occurs.

When using fwrite() for record output, set *size* to 1 and *count* to the length of the record to obtain the number of bytes written. You can only write one record at a time when using record I/O.

The value of errno may be set to:

Value Meaning

ENOTWRITE

The file is not open for write operations.

EPAD Padding occurred on a write operation.

EPUTANDGET

An illegal write operation occurred after a read operation.

ESTDERR

stderr cannot be opened.

ESTDIN

stdin cannot be opened.

ESTDOUT

stdout cannot be opened.

ETRUNC

Truncation occurred on I/O operation.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

Example that uses fwrite()

This example writes NUM long integers to a stream in binary format.

```
#include <stdio.h>
#define NUM 100
int main(void)
   FILE *stream;
  long list[NUM];
  int numwritten;
  int i;
   stream = fopen("MYLIB/MYFILE", "w+b");
   /* assign values to list[] */
  for (i=0; i \le NUM; i++)
      list[i]=i;
  numwritten = fwrite(list, sizeof(long), NUM, stream);
  printf("Number of items successfully written = %d\n", numwritten);
```

Related Information

- "fopen() Open Files" on page 92
- "fread() Read Items" on page 108
- "<stdio.h>" on page 15

fwscanf() — Read Data from Stream Using Wide Character

Format

```
#include <stdio.h>
#include <wchar.h>
int fwscanf(FILE *stream, const wchar_t *format, argument-list);
```

Language Level: ANSI

Thread Safe: YES

Description

The fwscanf() function reads input from the stream pointed to by stream, under control of the wide string pointed to by format. The format string specifies the admissible input sequences and how they are to be converted for assignment. To receive the converted input, the fwscanf() function uses subsequent arguments as pointers to the objects.

Each argument in argument-list must point to a variable with a type that corresponds to a type specifier in format.

If insufficient arguments exist for the format, the behavior is undefined. If the format is exhausted while arguments remain, the fwscanf() function evaluates the excess arguments, but otherwise ignores them.

The format consists of zero or more directives: one or more white-space wide characters; an ordinary wide character (neither % nor a white-space wide character); or a conversion specification. Each conversion specification is introduced by a %.

The format has the same form and function as the format string for the scanf()function, with the following exceptions:

- %c (with no l prefix) converts one or more wchar_t characters (depending on precision) to multibyte characters, as if by calling wcrtomb().
- %lc and %C convert one or more wchar_t characters (depending on precision) to an array of wchar_t.
- %s (with no 1 prefix) converts a sequence of non-white-space wchar t characters to multibyte characters, as if by calling the wcrtomb() function. The array includes the ending null character.
- %ls and %S copy an array of wchar_t, including the ending null wide character, to an array of wchar_t.

If the data is from stdin, and stdin has not been overridden, the data is assumed to be in the CCSID of the job. The data is converted as required by the format specifications. If the file that is being read is not opened with file mode rb, then invalid conversion will occur as there are no valid wide job CCSIDs.

If a conversion specification is invalid, the behavior is undefined. If the fwscanf() function encounters end-of-file during input, conversion is ended. If end-of-file occurs before the fwscanf() function reads any characters matching the current directive (other than leading white space, where permitted), execution of the current directive ends with an input failure. Otherwise, unless execution of the current directive terminates with a matching failure, execution of the following directive (other than %n, if any) ends with an input failure.

The fwscanf() function leaves trailing white space (including new-line wide characters) unread, unless matched by a directive. You cannot determine the success of literal matches and suppressed assignments other than through the %n directive.

Note: This function is available when SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) are specified on the compilation command.

Return Value

The fwscanf() function returns the number of input items assigned, which can be fewer than provided for, in the event of an early matching failure.

If an input failure occurs before any conversion, the fwscanf() function returns EOF.

Example that uses fwscanf()

This example opens the file myfile.dat for input, and then scans this file for a string, a long integer value, a character, and a float...

```
#include <stdio.h>
#include <wchar.h>
#define MAX_LEN
                     80
int main(void)
  FILE *stream;
  long 1;
  float fp;
  char s[MAX LEN+1];
  char c;
  stream = fopen("myfile.dat", "r");
   /* Read data from file. */
  fwscanf(stream, L"%s", &s[0]);
  fwscanf(stream, L"%ld", &1);
  fwscanf(stream, L"%c", &c);
fwscanf(stream, L"%f", &fp);
  printf("string = %s\n", s);
  printf("long integer = %ld\n", 1);
  printf("char = %c\n", c);
  printf("float = %f\n", fp);
  return 0;
  /**************
     If myfile.dat contains:
     abcdefghijklmnopqrstuvwxyz 343.2.
     The output should be:
     string = abcdefghijklmnopqrstuvwxyz
     long integer = 343
     char = .
     float = 2.000000
```

UNICODE example that uses fwscanf()

This example reads a UNICODE string from unicode.dat and prints it to the screen. The example is compiled with LOCALETYPE(*LOCALEUCS2) SYSIFCOPT(*IFSIO):

```
void main(void)
FILE *stream;
wchar t buffer[20];
setlocale(LC UCS2 ALL,"");
stream=fopen("unicode.dat","rb");
fwscanf(stream,L"%1s", buffer);
wprintf(L"The string read was :%ls\n",buffer);
fclose(stream);
/* If the input in unicode.dat is :
and ABC is in unicode which in hex would be 0x0041, 0x0042, 0x0043
then the output will be similiar to:
The string read was :ABC
```

- "fscanf() Read Formatted Data" on page 113
- "fwprintf() Format Data as Wide Characters and Write to a Stream" on page 123
- "scanf() Read Data" on page 299
- "<stdio.h>" on page 15
- "swprintf() Format and Write Wide Characters to Buffer" on page 367
- "swscanf() Read Wide Character Data" on page 369
- "wscanf() Read Data Using Wide-Character Format String" on page 448
- "<wchar.h>" on page 18

gamma() — Gamma Function

Format

```
#include <math.h>
double gamma (double x);
```

Language Level: ILE C Extension

Description

The gamma () function computes the natural logarithm of the absolute value of G(x) $(\ln(|G(x)|))$, where

$$G(x) = \int_{0}^{\infty} e^{-t} \times t^{x-1} dt$$

The argument x must be a positive real value.

Return Value

The gamma() function returns the value of ln(|G(x)|). If x is a negative value, errno is set to EDOM. If the result causes an overflow, gamma() returns HUGE_VAL and sets errno to ERANGE.

Example that uses gamma()

This example uses gamma() to calculate ln(|G(x)|), where x = 42.

```
#include <math.h>
#include <stdio.h>
int main(void)
  double x=42, g_at_x;
  g_at_x = exp(gamma(x)); /* g_at_x = 3.345253e+49 */
  printf ("The value of G(%4.21f) is %7.2e\n", x, g_at_x);
/******************** Output should be similar to: ********
The value of G(42.00) is 3.35e+49
```

Related Information

- "Bessel Functions" on page 51
- "erf() erfc() Calculate Error Functions" on page 72
- "<math.h>" on page 8

gcvt - Convert Floating-Point to String

Format

```
#include <stdlib.h>
char * gcvt(double value, int ndec, char *buffer);
```

Note: The gcvt function is supported only for C++, not for C.

Language Level: Extension

Description

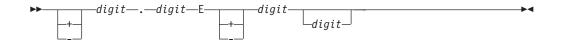
_gcvt() converts a floating-point value to a character string pointed to by *buffer*. The buffer should be large enough to hold the converted value and a null character (\0) that _gcvt() automatically adds to the end of the string. There is no provision for overflow.

_gcvt() attempts to produce *ndec* significant digits in FORTRAN F format. Failing that, it produces ndec significant digits in FORTRAN E format. Trailing zeros might be suppressed in the conversion if they are insignificant.

A FORTRAN F number has the following format:



A FORTRAN E number has the following format:



_gcvt also converts NaN and infinity values to the strings NAN and INFINITY, respectively.

Return Value

_gcvt() returns a pointer to the string of digits. If it cannot allocate memory to perform the conversion, _gcvt() returns an empty string and sets errno to ENOMEM.

Example that uses _gcvt()

This example converts the value -3.1415e3 to a character string and places it in the character array buffer1.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    char buffer1[10];
    _gcvt(-3.1415e3, 7, buffer1);
    printf("The first result is %s \n", buffer1);
    return 0;
}
```

The output should be:

The first result is -3141.5

Related Information:

- · Infinity and NaN Support
- <stdlib.h>

getc() - getchar() - Read a Character

Format

```
#include <stdio.h>
int getc(FILE *stream);
int getchar(void);
```

Language Level: ANSI

Thread Safe: NO. #undef getc or #undef getchar allows the getc or getchar function to be called instead of the macro version of these functions. The functions are thread safe.

Description

The getc() function reads a single character from the current *stream* position and advances the *stream* position to the next character. The getchar() function is identical to getc(stdin).

The difference between the getc() and fgetc() functions is that getc() can be implemented so that its arguments can be evaluated multiple times. Therefore, the stream argument to getc() should not be an expression with side effects.

Return Value

The getc() and getchar() functions return the character read. A return value of EOF indicates an error or end-of-file condition. Use ferror() or feof() to determine whether an error or an end-of-file condition occurred.

The value of errno may be set to:

Value Meaning

EBADF

The file pointer or descriptor is not valid.

EGETANDPUT

An illegal read operation occurred after a write operation.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

The getc() and getchar() functions are not supported in record mode.

Example that uses getc()

This example gets a line of input from the stdin stream. You can also use getc(stdin) instead of getchar() in the for statement to get a line of input from stdin.

```
#include <stdio.h>
#define LINE 80
int main(void)
 char buffer[LINE+1];
 int i;
 int ch;
 printf( "Please enter string\n" );
  /* Keep reading until either:
     1. the length of LINE is exceeded or
     2. the input character is EOF or
    3. the input character is a new-line character
 for ( i = 0; ( i < LINE ) && (( ch = getchar()) != EOF) && ( ch !='\n' ): ++i )
              ( ch !='\n' ); ++i )
   buffer[i] = ch;
 buffer[i] = '\0'; /* a string should always end with '\0'! */
 printf( "The string is %s\n", buffer );
```

Related Information

• "fgetc() — Read a Character" on page 82

- "fgetwc() Read Wide Character from Stream" on page 86
- "gets() Read a Line" on page 137
- "getwc() Read Wide Character from Stream" on page 138
- "getwchar() Get Wide Character from stdin" on page 140
- "putc() putchar() Write a Character" on page 210
- "ungetc() Push Character onto Input Stream" on page 381
- "<stdio.h>" on page 15

getenv() — Search for Environment Variables

Format

```
#include <stdlib.h>
char *getenv(const char *varname);
```

Language Level: ANSI

Description

The getenv() function searches the list of environment variables for an entry corresponding to *varname*.

Return Value

The getenv() function returns a pointer to the string containing the value for the specified *varname* in the current environment. If getenv() cannot find the environment string, NULL is returned, and errno is set to indicate the error.

Example that uses getenv()

```
#include <stdlib.h>
#include <stdlib.h>
#include <stdio.h>

/* Where the environment variable 'PATH' is set to a value. */
int main(void)
{
   char *pathvar;
   pathvar = getenv("PATH");
   printf("pathvar=%s",pathvar);
}
```

Related Information

- "<stdlib.h>" on page 16
- "putenv() Change/Add Environment Variables" on page 211
- Environment Variable See the API topic

_GetExcData() — Get Exception Data

Format

```
#include <signal.h>
void _GetExcData(_INTRPT_Hndlr_Parms_T *parms);
```

Language Level: ILE C Extension

Description

The GetExcData() function returns information about the current exception from within a C signal handler. The caller of the GetExcData() function must allocate enough storage for a structure of type _INTRPT_Hndlr_Parms_T. If the GetExcData() function is called from outside a signal handler, the storage pointed to by parms is not updated.

This function is not available when SYSIFCOPT(*ASYNCSIGNAL) is specified on the compilation commands. When SYSIFCOPT(*ASYNCSIGNAL) is specified, a signal handler established with the ILE C signal() function has no way to access any exception information that may have caused the signal handler to be invoked. An extended signal handler established with the OS/400[®] signation() function, however, does have access to this exception information. The extended signal handler has the following function prototype:

```
void func( int signo, siginfo t *info, void *context )
```

The exception information is appended to the siginfo t structure, which is then passed as the second parameter to the extended signal handler.

The siginfo t structure is defined in signal.h. The exception-related data follows the si sigdata field in the siginfo tstructure. You can address it from the se data field of the sigdata t structure.

The format of the exception data appended to the siginfo t structure is defined by the _INTRPT_Hndlr_Parms_T structure in except.h.

Return Value

There is no return value.

Example that uses _GetExcData()

This example shows how exceptions from MI library functions can be monitored and handled using a signal handling function. The signal handler my_signal_handler is registered before the rslvsp() function signals a 0x2201 exception. When a SIGSEGV signal is raised, the signal handler is called. If an 0x2201 exception occurred, the signal handler calls the QUSRCRTS API to create a space.

```
#include <signal.h>
#include <QSYSINC/MIH/RSLVSP>
#include <QSYSINC/H/QUSCRTUS>
#include <string.h>
#define CREATION SIZE 65500
void my signal handler(int sig) {
   _INTRPT_Hndlr_Parms_T excp_data;
                        error code = 0;
   /* Check the message id for exception 0x2201 */
   _GetExcData(&excp_data);
   if (!memcmp(excp data.Msg Id, "MCH3401", 7))
      QUSCRTUS("MYSPACE QTEMP
               "MYSPACE
               CREATION SIZE,
               "\0",
               "*ALL
               "MYSPACE example for Programmer's Reference
               "*YES
               &error code);
}
```

- "signal() Handle Interrupt Signals" on page 313
- "<except.h>" on page 4

gets() — Read a Line

Format

```
#include <stdio.h>
char *gets(char *buffer);
```

Language Level: ANSI

Thread Safe: YES

Description

The gets() function reads a line from the standard input stream stdin and stores it in *buffer*. The line consists of all characters up to but not including the first new-line character (\n) or EOF. The gets() function then replaces the new-line character, if read, with a null character (\n 0) before returning the line.

Return Value

If successful, the gets() function returns its argument. A NULL pointer return value indicates an error, or an end-of-file condition with no characters read. Use the ferror() function or the feof() function to determine which of these conditions occurred. If there is an error, the value that is stored in *buffer* is undefined. If an end-of-file condition occurs, *buffer* is not changed.

Example that uses gets()

This example gets a line of input from stdin.

```
#include <stdio.h>
#define MAX LINE 100
int main(void)
  char line[MAX LINE];
  char *result;
   printf("Please enter a string:\n");
   if ((result = gets(line)) != NULL)
     printf("The string is: %s\n", line);
   else if (ferror(stdin))
     perror("Error");
```

- "fgets() Read a String" on page 85
- "fgetws() Read Wide-Character String from Stream" on page 88
- "feof() Test End-of-File Indicator" on page 79
- "ferror() Test for Read/Write Errors" on page 79
- "fputs() Write String" on page 103
- "getc() getchar() Read a Character" on page 133
- "puts() Write a String" on page 212
- "<stdio.h>" on page 15

getwc() — Read Wide Character from Stream

Format

```
#include <stdio.h>
#include <wchar.h>
wint_t getwc(FILE *stream);
```

Language Level: ANSI

Thread Safe: YES

Description

The getwc() function reads the next multibyte character from stream, converts it to a wide character, and advances the associated file position indicator for stream.

The getwc() function is equivalent to the fgetwc() function except that, if it is implemented as a macro, it can evaluate stream more than once. Therefore, the argument should never be an expression with side effects.

The behavior of the getwc() function is affected by the LC_CTYPE category of the current locale. If you change the category between subsequent read operations on the same stream, undefined results can occur. Using non-wide-character functions with the getwc() function on the same stream results in undefined behavior.

After calling the getwc() function, flush the buffer or reposition the stream pointer before calling a write function for the stream, unless EOF has been reached. After a write operation on the stream, flush the buffer or reposition the stream pointer before calling the getwc() function.

Note: This function is available when SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) are specified on the compilation command.

Return Value

The getwc() function returns the next wide character from the input stream, or WEOF. If an error occurs, the getwc() function sets the error indicator. If the getwc() function encounters the end-of-file, it sets the EOF indicator. If an encoding error occurs during conversion of the multibyte character, the getwc() function sets errno to EILSEQ.

Use the ferror() or feof() functions to determine whether an error or an EOF condition occurred. EOF is only reached when an attempt is made to read past the last byte of data. Reading up to and including the last byte of data does not turn on the EOF indicator.

Example that uses getwc()

```
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <errno.h>
int main(void)
  FILE *stream;
  wint t wc;
  if (NULL == (stream = fopen("getwc.dat", "r"))) {
     printf("Unable to open: \"getwc.dat\"\n");
     exit(1);
  errno = 0;
  while (WEOF != (wc = getwc(stream)))
     printf("wc = %lc\n", wc);
  if (EILSEQ == errno) {
     printf("An invalid wide character was encountered.\n");
     exit(1);
  fclose(stream);
  return 0;
  /*********************
     Assuming the file getwc.dat contains:
     Hello world!
     The output should be similar to:
     wc = H
     wc = e
     wc = 1
     wc = 1
     WC = 0
  *************************************
```

Related Information

- "fgetwc() Read Wide Character from Stream" on page 86
- "getwchar() Get Wide Character from stdin"
- "getc() getchar() Read a Character" on page 133
- "putwc() Write Wide Character" on page 213
- "<stdio.h>" on page 15
- "ungetwc() Push Wide Character onto Input Stream" on page 382
- "<wchar.h>" on page 18

getwchar() — Get Wide Character from stdin

Format

#include <wchar.h> wint t getwchar(void);

Language Level: ANSI

Thread Safe: YES

Description

The getwchar() function reads the next multibyte character from stdin, converts it to a wide character, and advances the associated file position indicator for stdin. A call to the getwchar() function is equivalent to a call to getwc(stdin).

The behavior of the getwchar() function is affected by the LC CTYPE category of the current locale. If you change the category between subsequent read operations on the same stream, undefined results can occur. Using non-wide-character functions with the getwchar() function on stdin results in undefined behavior.

Note: This function is available when SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) are specified on the compilation command.

Return Value

The getwchar() function returns the next wide character from stdin or WEOF. If the getwchar() function encounters EOF, it sets the EOF indicator for the stream and returns WEOF. If a read error occurs, the error indicator for the stream is set, and the getwchar() function returns WEOF. If an encoding error occurs during the conversion of the multibyte character to a wide character, the getwchar() function sets errno to EILSEQ and returns WEOF.

Use the ferror() or feof() functions to determine whether an error or an EOF condition occurred. EOF is only reached when an attempt is made to read past the last byte of data. Reading up to and including the last byte of data does not turn on the EOF indicator.

Example that uses getwchar()

This example uses the getwchar() to read wide characters from the keyboard, then prints the wide characters.

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
int main(void)
  wint t wc;
  errno = 0;
  while (WEOF != (wc = getwchar()))
    printf("wc = %lc\n", wc);
  if (EILSEQ == errno) {
    printf("An invalid wide character was encountered.\n");
    exit(1);
  return 0;
  /***********************
                             (note: ^Z is CNTRL-Z)
    Assuming you enter: abcde Z
    The output should be:
    wc = a
    wc = b
    WC = C
    wc = d
    wc = e
```

- "fgetc() Read a Character" on page 82
- "fgetwc() Read Wide Character from Stream" on page 86
- "fgetws() Read Wide-Character String from Stream" on page 88
- "getc() getchar() Read a Character" on page 133
- "getwc() Read Wide Character from Stream" on page 138
- "ungetwc() Push Wide Character onto Input Stream" on page 382
- "<wchar.h>" on page 18

gmtime() — Convert Time

Format

```
#include <time.h>
struct tm *gmtime(const time_t *time);
```

Language Level: ANSI

Thread Safe: NO. Use gmtime r() instead.

Description

The gmtime() function breaks down the time value, in seconds, and stores it in a tm structure, defined in <time.h>. The value time is usually obtained by a call to the time() function.

The fields of the tm structure include:

```
tm_sec
       Seconds (0-61)
tm_min
       Minutes (0-59)
tm hour
       Hours (0-23)
tm_mday
       Day of month (1-31)
tm_mon
       Month (0-11; January = 0)
tm_year
       Year (current year minus 1900)
tm_wday
       Day of week (0-6; Sunday = 0)
tm_yday
       Day of year (0-365; January 1 = 0)
tm isdst
       Zero if Daylight Saving Time is not in effect; positive if Daylight Saving
```

Return Value

The gmtime() function returns a pointer to the resulting tm structure.

Time is in effect; negative if the information is not available.

Note:

- The range (0-61) for tm_sec allows for as many as two leap seconds.
- The gmtime() and localtime() functions may use a common, statically allocated buffer for the conversion. Each call to one of these functions may alter the result of the previous call.
- Calendar time is the number of seconds that have elapsed since EPOCH, which is 00:00:00, January 1, 1970 Universal Coordinate Time (UTC).
- The time() function uses the QUTCOFFSET system value for calculating UTC. The QUTCOFFSET value specifies the difference in hours and minutes between UTC, also known as Greenwich mean time, and the current system time. You can override the QUTCOFFSET value by specifying the TZDIFF and TZNAME category of the current locale.

Note: This function is locale sensitive.

Example that uses gmtime()

This example uses the gmtime() function to adjust a time_t representation to a Coordinated Universal Time character string, and then converts it to a printable string using the asctime() function.

```
#include <stdio.h>
#include <time.h>
int main(void)
  time_t ltime;
  time(localtime());
  printf ("Coordinated Universal Time is %s\n",
           asctime(gmtime(localtime())));
/******************* Output should be similar to: ********
Coordinated Universal Time is Wed Aug 18 21:01:44 1993
```

- "asctime() Convert Time to Character String" on page 40
- "asctime_r() Convert Time to Character String (Restartable)" on page 42
- "ctime() Convert Time to Character String" on page 66
- "ctime_r() Convert Time to Character String (Restartable)" on page 67
- "gmtime_r() Convert Time (Restartable)"
- "localtime() Convert Time" on page 163
- "localtime_r() Convert Time (Restartable)" on page 164
- "mktime() Convert Local Time" on page 193
- "setlocale() Set Locale" on page 308
- "time() Determine Current Time" on page 373
- "<time.h>" on page 17

gmtime_r() — Convert Time (Restartable)

Format

```
#include <time.h>
struct tm *gmtime_r(const time_t *time, struct tm *result);
```

Description

This function is the restartable version of gmtime().

The gmtime r() function breaks down the time value, in seconds, and stores it in a tm structure, defined in <time.h>. The value time is usually obtained by a call to the time() function.

The fields of the tm structure include:

```
tm_sec
       Seconds (0-61)
tm min
       Minutes (0-59)
tm_hour
       Hours (0-23)
tm_mday
       Day of month (1-31)
```

```
tm_mon
```

Month (0-11; January = 0)

tm_year

Year (current year minus 1900)

tm_wday

Day of week (0-6; Sunday = 0)

tm_yday

Day of year (0-365; January 1 = 0)

tm_isdst

Zero if Daylight Saving Time is not in effect; positive if Daylight Saving Time is in effect; negative if the information is not available.

Return Value

The gmtime r() function returns a pointer to the resulting tm structure.

Note:

- The range (0-61) for tm_sec allows for as many as two leap seconds.
- The gmtime() and localtime() functions may use a common, statically allocated buffer for the conversion. Each call to one of these functions may alter the result of the previous call. The asctime r(), ctime r(), gmtime r(), and localtime r() functions do not use a common, statically-allocated buffer to hold the return string. These functions can be used in the place of the asctime(), ctime(), gmtime(), and localtime() functions if reentrancy is desired.
- Calendar time is the number of seconds that have elapsed since EPOCH, which is 00:00:00, January 1, 1970 Universal Coordinate Time (UTC).
- The time() function uses the QUTCOFFSET system value for calculating UTC. The QUTCOFFSET value specifies the difference in hours and minutes between UTC, also known as Greenwich mean time, and the current system time. You can override the QUTCOFFSET value by specifying the TZDIFF and TZNAME category of the current locale.

Example that uses gmtime_r()

This example uses the gmtime r() function to adjust a time_t representation to a Coordinated Universal Time character string, and then converts it to a printable string using the asctime_r() function.

```
#include <stdio.h>
#include <time.h>
int main(void)
  time t ltime;
  struct tm mytime;
  char buf[50];
   time(localtime());
  printf ("Coordinated Universal Time is %s\n",
           asctime r(gmtime r(localtime(), &mytime), buf));
/************************ Output should be similar to: ********
Coordinated Universal Time is Wed Aug 18 21:01:44 1993
```

- "asctime() Convert Time to Character String" on page 40
- "asctime_r() Convert Time to Character String (Restartable)" on page 42
- "ctime() Convert Time to Character String" on page 66
- "ctime_r() Convert Time to Character String (Restartable)" on page 67
- "gmtime() Convert Time" on page 141
- "localtime() Convert Time" on page 163
- "localtime_r() Convert Time (Restartable)" on page 164
- "mktime() Convert Local Time" on page 193
- "time() Determine Current Time" on page 373
- "<time.h>" on page 17

hypot() — Calculate Hypotenuse

Format

```
#include <math.h>
double hypot(double side1, double side2);
```

Language Level: ILE C Extension

Description

The hypot() function calculates the length of the hypotenuse of a right-angled triangle based on the lengths of two sides *side1* and *side2*. A call to the hypot() function is equivalent to:

```
sqrt(side1 * side1 + side2 * side2);
```

Return Value

The hypot() function returns the length of the hypotenuse. If an overflow results, hypot() sets errno to ERANGE and returns the value HUGE_VAL. If an underflow results, hypot() sets errno to ERANGE and returns zero. The value of errno may also be set to EDOM.

Example that uses hypot()

This example calculates the hypotenuse of a right-angled triangle with sides of 3.0 and 4.0.

```
#include <math.h>
int main(void)
  double x, y, z;
  x = 3.0;
  y = 4.0;
  z = hypot(x,y);
  printf("The hypotenuse of the triangle with sides %lf and %lf"
         " is %lf\n", x, y, z);
/******* Output should be similar to: ********
The hypotenuse of the triangle with sides 3.000000 and 4.000000 is 5.000000
```

Related Information

- "sqrt() Calculate Square Root" on page 318
- "<math.h>" on page 8

isascii() — Test for ASCII Value

Format

```
#include <ctype.h>
int isascii(int c);
```

Language Level: XPG4

Description

The isascii() function tests if an EBCDIC character, in the current locale, is a valid 7-bit US-ASCII character.

Return Value

The isascii() function returns nonzero if c is the EBCDIC encoding, in the current locale, for a character in the 7-bit US-ASCII character set. Otherwise, it returns 0.

Example that uses isascii()

This example tests the integers from 0x7c to 0x82, and prints the corresponding ASCII character if the integer is within the ASCII range.

```
#include <stdio.h>
#include <ctype.h>
int main(void)
  int ch;
  for (ch = 0x7c; ch <= 0x82; ch++) {
     printf("%#04x
                    ", ch);
     if (isascii(ch))
        printf("The ASCII character is %c\n", ch);
        printf("Not an ASCII character\n");
  return 0;
  /**************
     The output should be:
0x7c
       The ASCII character is @
0x7d
       The ASCII character is '
0x7e
      The ASCII character is =
0x7f The ASCII character is "
0x80 Not an ASCII character
0x81
       The ASCII character is a
       The ASCII character is b
0x82
  ************************************
```

- "isalnum() isxdigit() Test Integer Value"
- "iswalnum() to iswxdigit() Test Wide Integer Value" on page 150
- "toascii() Convert Character" on page 376
- "tolower() toupper() Convert Character Case" on page 377
- "towlower() –towupper() Convert Wide Character Case" on page 379
- "<ctype.h>" on page 3

isalnum() - isxdigit() — Test Integer Value

Format

```
#include <ctype.h>
int isalnum(int c);
/* Test for upper- or lowercase letters, or decimal digit */
int isalpha(int c);
/* Test for alphabetic character */
int iscntrl(int c);
/* Test for any control character */
int isdigit(int c);
/* Test for decimal digit */
int isgraph(int c);
/* Test for printable character excluding space */
int islower(int c);
/* Test for lowercase */
int isprint(int c);
/* Test for printable character including space */
int ispunct(int c);
/* Test for any nonalphanumeric printable character */
/* excluding space */
int isspace(int c);
/* Test for whitespace character */
int isupper(int c);
```

```
/* Test for uppercase */
int isxdigit(int c);
/* Test for hexadecimal digit */
```

Language Level: ANSI

Description

The <ctype.h> functions listed test a character with an integer value. The LC_CTYPE category of the active locale determines the results returned from these functions.

Return Value

These functions return a nonzero value if the integer satisfies the test condition, or a zero value if it does not. The integer variable c must be representable as an unsigned char.

Note: EOF is a valid input value.

Example that uses <ctype.h> functions

This example analyzes all characters between code 0x0 and code UPPER_LIMIT, printing A for alphabetic characters, AN for alphanumerics, U for uppercase, L for lowercase, D for digits, X for hexadecimal digits, S for spaces, PU for punctuation, PR for printable characters, G for graphics characters, and C for control characters. This example prints the code if printable.

The output of this example is a 256-line table showing the characters from 0 to 255 that possess the attributes tested.

- "tolower() toupper() Convert Character Case" on page 377
- "<ctype.h>" on page 3
- "isblank() Test for Blank or Tab Character"

isblank() — Test for Blank or Tab Character

Format

```
#include <ctype.h>
int isblank(int c);
```

Note: The isblank() function is supported only for C++, not for C.

Thread Safe: YES.

Description

The isblank() function tests if a character is either the EBCDIC space or EBCDIC tab character.

Return Value

The isblank() function returns nonzero if c is either the EBCDIC space character or the EBCDIC tab character, otherwise it returns 0.

Example that uses isblank()

This example tests several characters using isblank().

```
#include <stdio.h>
#include <ctype.h>
int main(void)
  char *buf = "a b\tc";
  int i;
  for (i = 0; i < 5; i++) {
    if (isblank(buf[i]))
       printf("Character %d is not a blank.\n", i);
       printf("Character %d is a blank\n", i);
  }
  return 0;
/***********
    The output should be
Character 0 is not a blank.
Character 1 is a blank.
Character 2 is not a blank.
Character 3 is a blank.
Character 4 is not a blank.
*****************************
```

- "isalnum() isxdigit() Test Integer Value" on page 147
- "iswalnum() to iswxdigit() Test Wide Integer Value"
- "isascii() Test for ASCII Value" on page 146
- "tolower() toupper() Convert Character Case" on page 377
- "towlower() –towupper() Convert Wide Character Case" on page 379
- "<ctype.h>" on page 3

iswalnum() to iswxdigit() — Test Wide Integer Value

Format

```
#include <wctype.h>
int iswalnum(wint_t wc);
int iswalpha(wint_t wc);
int iswcntrl(wint_t wc);
int iswdigit(wint_t wc);
int iswgraph(wint_t wc);
int iswlower(wint_t wc);
int iswprint(wint_t wc);
int iswpunct(wint_t wc);
int iswspace(wint_t wc);
int iswspace(wint_t wc);
int iswupper(wint_t wc);
int iswxdigit(wint_t wc);
```

Thread Safe: YES.

Description

The functions listed above, which are all declared in <wctype.h>, test a given wide integer value.

The value of wc must be a wide-character code corresponding to a valid character in the current locale, or must equal the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

If the program is compiled with LOCALETYPE(*LOCALEUCS2), the wide character properties are those defined by the LC_UCS2_CTYPE category of the current locale.

Here are descriptions of each function in this group.

iswalnum()

Test for a wide alphanumeric character.

iswalpha()

Test for a wide alphabetic character, as defined in the alpha class of the LC_CTYPE category of the current locale.

iswcntrl()

Test for a wide control character, as defined in the cntrl class of the LC_CTYPE category of the current locale.

iswdigit()

Test for a wide decimal-digit character: 0 through 9, as defined in the digit class of the LC_CTYPE category of the current locale.

iswgraph()

Test for a wide printing character, not a space, as defined in the graph class of the LC_CTYPE category of the current locale.

iswlower()

Test for a wide lowercase character, as defined in the lower class of the LC_CTYPE category of the current locale or for which none of the iswcntrl(), iswdigit(), iswspace() function are true.

iswprint()

Test for any wide printing character, as defined in the print class of the LC_CTYPE category of the current locale.

iswpunct()

Test for a wide non-alphanumeric, non-space character, as defined in the punct class of the LC_CTYPE category of the current locale.

iswspace()

Test for a wide whitespace character, as defined in the space class of the LC_CTYPE category of the current locale.

iswupper()

Test for a wide uppercase character, as defined in the upper class of the LC_CTYPE category of the current locale.

iswxdigit()

Test for a wide hexadecimal digit 0 through 9, a through f, or A through F. as defined in the xdigit class of the LC_CTYPE category of the current locale.

The LC CTYPE or LC UCS2 TYPE category of the active locale determines the results returned from these functions.

Note: The <wctype.h> functions are accessible only when LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

Returned Value

These functions return a nonzero value if the wide integer satisfies the test value, or a 0 value if it does not. The value for wc must be representable as a wide unsigned char. WEOF is a valid input value.

The behavior of these wide-character functions is affected by the LC_CTYPE or LC_UCS2_TYPE category of the current locale. If you change the category, undefined results can occur.

Example

```
#include <stdio.h>
#include <wctype.h>
int main(void)
    int wc;
    for (wc=0; wc \le 0xFF; wc++) {
        printf("%3d", wc);
        printf(" %#4x ", wc);
        printf("%3s", iswalnum(wc) ? "AN" :
        printf("%2s", iswalpha(wc) ? "A"
printf("%2s", iswcntrl(wc) ? "C"
printf("%2s", iswdigit(wc) ? "D"
printf("%2s", iswgraph(wc) ? "G"
        printf("%2s", iswlower(wc) ? "L"
        printf(" %c", iswprint(wc) ? wc
        printf("%3s", iswpunct(wc)
                                              ? "PU"
        printf("%2s", iswspace(wc)
                                              ? "$"
        printf("%3s", iswprint(wc) ? "PR'
printf("%2s", iswupper(wc) ? "U"
printf("%2s", iswxdigit(wc) ? "X"
                                               ? "PR"
        putchar('\n');
```

Related Information

"<wctype.h>" on page 18

iswctype() — Test for Character Property

Format

```
#include <wctype.h>
int iswctype(wint_t wc, wctype_t wc_prop);
```

Thread Safe: YES.

Description

The iswctype() function determines whether the wide character wc has the property wc_prop. If the value of wc is neither WEOF nor any value of the wide characters that corresponds to a multibyte character, the behavior is undefined. If the value of wc_prop is incorrect (that is, it is not obtained by a previous call to the wctype() function, or wc_prop has been invalidated by a subsequent call to the setlocale() function that has affected category LC_CTYPE or LC_UCS2_TYPE), the behavior is undefined.

If the program is compiled with LOCALETYPE(*LOCALEUCS2), the wide character properties are those defined by the LC_UCS2_CTYPE category of the current locale.

Note: This function is accessible only when LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

Return Value

The iswctype() function returns true if the value of the wide character wc has the property wc_prop .

The following strings, alnum through to xdigit are reserved for the standard character classes. The functions are shown as follows with their equivalent isw*() function:

```
/* is equivalent to */
iswctype(wc, wctype("alnum"));
                                                                      iswalnum(wc);
iswctype(wc, wctype("alpha"));
                                    /* is equivalent to */
                                                                      iswalpha(wc);
iswctype(wc, wctype("cntrl"));
                                    /* is equivalent to */
                                                                      iswcntrl(wc);
                                     /* is equivalent to */
iswctype(wc, wctype("digit"));
                                                                      iswdigit(wc);
                                   /* is equivalent to */
iswctype(wc, wctype("graph"));
                                                                     iswgraph(wc);
iswctype(wc, wctype("lower"));
                                   /* is equivalent to */
                                                                     iswlower(wc);
iswctype(wc, wctype("print"));
                                    /* is equivalent to */
                                                                     iswprint(wc);
                                   /* is equivalent to */
iswctype(wc, wctype("punct"));
                                                                     iswpunct(wc);
iswctype(wc,wctype("space"));
                                   /* is equivalent to */
                                                                     iswspace(wc);
iswctype(wc, wctype("upper"));
                                   /* is equivalent to */
                                                                     iswupper(wc);
                                   /* is equivalent to */
iswctype(wc, wctype("xdigit"));
                                                                     iswxdigit(wc);
```

Example that uses iswctype()

```
#include <stdio.h>
#include <wctype.h>
int main(void)
   int wc;
   for (wc=0; wc \leftarrow 0xFF; wc++) {
       printf("%3d", wc);
       printf(" %#4x ", wc);
       printf("%3s", iswctype(wc, wctype("alnum")) ? "AN" : " ");
       printf("%2s", iswctype(wc, wctype("alpha")) ? "A" : " ");
       printf("%2s", iswctype(wc, wctype("cntrl")) ? "C" : " ");
       printf("%2s", iswctype(wc, wctype("digit")) ? "D" : " ");
       printf("%2s", iswctype(wc, wctype("graph")) ? "G" : " ");
       printf("%2s", iswctype(wc, wctype("lower")) ? "L" : " ");
       printf(" %c", iswctype(wc, wctype("print")) ? wc
                                                               ? "PU" : " ");
       printf("%3s", iswctype(wc, wctype("punct"))
       printf("%2s", iswctype(wc, wctype("space")) ? "S" : " ");
printf("%3s", iswctype(wc, wctype("print")) ? "PR" : " ");
printf("%2s", iswctype(wc, wctype("upper")) ? "U" : " ");
printf("%2s", iswctype(wc, wctype("xdigit")) ? "X" : " ");
                                                               ? "S" : " ");
       putchar('\n');
}
```

Related Information

- "wctype() Get Handle for Character Property Classification" on page 438
- "iswalnum() to iswxdigit() Test Wide Integer Value" on page 150

_itoa - Convert Integer to String

Format

```
#include <stdlib.h>
char * itoa(int value, char *string, int radix);
```

Note: The itoa function is supported only for C++, not for C.

Language Level: Extension

Description

_itoa() converts the digits of the given value to a character string that ends with a null character and stores the result in *string*. The *radix* argument specifies the base of value; it must be in the range 2 to 36. If radix equals 10 and value is negative, the first character of the stored string is the minus sign (-).

Note: The space reserved for *string* must be large enough to hold the returned string. The function can return up to 33 bytes including the null character $(\setminus 0)$.

Return Value

itoa returns a pointer to *string*. There is no error return value.

Example that uses _itoa()

This example converts the integer value -255 to a decimal, a binary, and a hex number, storing its character representation in the array buffer.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
  char buffer[35];
  char *p;
  p = itoa(-255, buffer, 10);
  printf("The result of itoa(-255) with radix of 10 is sn", p;
  p = itoa(-255, buffer, 2);
  printf("The result of _itoa(-255) with radix of 2\n
                                                         is %s\n", p);
  p = _itoa(-255, buffer, 16);
  printf("The result of _itoa(-255) with radix of 16 is s\n", p);
   return 0;
```

The output should be:

```
The result of _itoa(-255) with radix of 10 is -255 The result of _itoa(-255) with radix of 2
     is 1111111111111111111111111100000001
The result of _itoa(-255) with radix of 16 is ffffff01
```

Related Information:

- "_gcvt Convert Floating-Point to String" on page 132
- "_ltoa Convert Long Integer to String" on page 167
- "_ultoa Convert Unsigned Long Integer to String" on page 380
- Integer to String

labs() — llabs() — Calculate Absolute Value of Long and Long Long Integer

```
Format (labs())
#include <stdlib.h>
long int labs(long int n);

Format (llabs())
#include <stdlib.h>
long long int llabs(long long int i);
```

Language Level: ANSI

Description

The labs() function produces the absolute value of its long integer argument n. The result may be undefined when the argument is equal to LONG_MIN, the smallest available long integer. The value LONG_MIN is defined in the limits.h> include file.

The llabs() function returns the absolute value of its long long integer operand. The result may be undefined when the argument is equal to LONG_LONG_MIN, the smallest available long integer. The value LONG_LONG_MIN is defined in the limits.h> include file.

Return Value

The labs() function returns the absolute value of n. There is no error return value.

The llabs() function returns the absolute value of i. There is no error return value.

Example that uses labs()

This example computes y as the absolute value of the long integer -41567.

Related Information

- "abs() Calculate Integer Absolute Value" on page 38
- "fabs() Calculate Floating-Point Absolute Value" on page 74

Idexp() — Multiply by a Power of Two

Format

```
#include <math.h>
double ldexp(double x, int exp);
```

Language Level: ANSI

Description

The ldexp() function calculates the value of $x * (2^{exp})$.

Return Value

The ldexp() function returns the value of $x^*(2^{exp})$. If an overflow results, the function returns +HUGE_VAL for a large result or -HUGE_VAL for a small result, and sets errno to ERANGE.

Example that uses | dexp()

This example computes y as 1.5 times 2 to the fifth power $(1.5*2^5)$:

```
#include <math.h>
#include <stdio.h>
int main(void)
  double x, y;
  int p;
  x = 1.5;
  p = 5;
  y = 1dexp(x,p);
  printf("%1f times 2 to the power of %d is %1f\n", x, p, y);
/******* Output should be similar to: ********
1.500000 times 2 to the power of 5 is 48.000000
```

Related Information

- "exp() Calculate Exponential Function" on page 73
- "frexp() Separate Floating-Point Value" on page 112
- "modf() Separate Floating-Point Value" on page 195
- "<math.h>" on page 8

Idiv() — Ildiv() — Perform Long and Long Long Division

```
Format (ldiv())
#include <stdlib.h>
ldiv_t ldiv(long int numerator, long int denominator);
Format (lldiv())
```

```
#include <stdlib.h>
lldiv t lldiv(long long int numerator, long long int denominator);
```

Language Level: ANSI

Thread Safe: YES. However, only the function version is thread safe. The macro version is NOT thread safe.

Description

The ldiv() function calculates the quotient and remainder of the division of numerator by denominator.

Return Value

The ldiv() function returns a structure of type ldiv_t, containing both the quotient (long int quot) and the remainder (long int rem). If the value cannot be represented, the return value is undefined. If denominator is 0, an exception is raised.

The 11div() subroutine computes the quotient and remainder of the numerator parameter by the denominator parameter.

The 11div() subroutine returns a structure of type lldiv_t, containing both the quotient and the remainder. The structure is defined as:

```
struct lldiv t
long long int quot; /* quotient */
long long int rem; /* remainder */
```

If the division is inexact, the sign of the resulting quotient is that of the algebraic quotient, and magnitude of the resulting quotient is the largest long long integer less than the magnitude of the algebraic quotient. If the result cannot be represented (for example, if the *denominator* is 0), the behavior is undefined.

Example that uses ldiv()

This example uses ldiv() to calculate the quotients and remainders for a set of two dividends and two divisors.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
  long int num[2] = \{45, -45\};
  long int den[2] = \{7,-7\};
  ldiv t ans; /* ldiv t is a struct type containing two long ints:
                 'quot' stores quotient; 'rem' stores remainder */
  short i,j;
  printf("Results of long division:\n");
  for (i = 0; i < 2; i++)
     for (j = 0; j < 2; j++)
       ans = ldiv(num[i], den[j]);
       printf("Dividend: %6ld Divisor: %6ld", num[i], den[j]);
       printf(" Quotient: %61d Remainder: %61d\n", ans.quot, ans.rem);
Results of long division:
Dividend: 45 Divisor: 7 Quotient: 6 Remainder:
Dividend: 45 Divisor: -7 Quotient: -6 Remainder: 3
Dividend: -45 Divisor: 7 Quotient: -6 Remainder: -3
Dividend: -45 Divisor: -7 Quotient: 6 Remainder: -3
```

- "div() Calculate Quotient and Remainder" on page 70
- "<stdlib.h>" on page 16

localeconv() — Retrieve Information from the Environment

Format

```
#include <locale.h>
struct lconv *localeconv(void);
```

Language Level: ANSI

Description

The localeconv() sets the components of a structure having type struct lconv to values appropriate for the current locale. The structure may be overwritten by another call to localeconv(), or by calling the setlocale() function and passing LC_ALL, LC_UCS2_ALL, LC_MONETARY, or LC_NUMERIC.

The structure contains the following elements (defaults shown are for the C locale):

Element	Purpose of Element	Default
char *decimal_point	Decimal-point character used to format non-monetary quantities.	
char *thousands_sep	Character used to separate groups of digits to the left of the decimal-point character in formatted non-monetary quantities.	1111

Element	Purpose of Element	Default
char *grouping	String indicating the size of each group of digits in formatted non-monetary quantities. Each character in the string specifies the number of digits in a group. The initial character represents the size of the group immediately to the left of the decimal delimiter. The characters following this define succeeding groups to the left of the previous group. If the last character is not UCHAR_MAX, the grouping is repeated using the last character as the size. If the last characater is UCHAR_MAX, grouping is only performed for the groups already in the string (no repetition). See Table 1 on page 160 for an example of how grouping works.	
char *int_curr_symbol	International currency symbol for the current locale. The first three characters contain the alphabetic international currency symbol. The fourth character (usually a space) is the character used to separate the international currency symbol from the monetary quantity.	"""
char *currency_symbol	Local currency symbol of the current locale.	""
char *mon_decimal_point	Decimal-point character used to format monetary quantities.	""
char *mon_thousands_sep	Separator for digits in formatted monetary quantities.	····
char *mon_grouping	String indicating the size of each group of digits in formatted monetary quantities. Each character in the string specifies the number of digits in a group. The initial character represents the size of the group immediately to the left of the decimal delimiter. The following characters define succeeding groups to the left of the previous group. If the last character is not UCHAR_MAX, the grouping is repeated using the last character as the size. If the last character is UCHAR_MAX, grouping is only performed for the groups already in the string (no repetition). See Table 1 on page 160 for an example of how grouping works.	
char *positive_sign	String indicating the positive sign used in monetary quantities.	""
char *negative_sign	String indicating the negative sign used in monetary quantities.	""

-

Element	Purpose of Element	Default
char int_frac_digits	The number of displayed digits to the right of the decimal place for internationally formatted monetary quantities.	UCHAR_MAX
char frac_digits	Number of digits to the right of the decimal place in monetary quantities.	UCHAR_MAX
char p_cs_precedes	1 if the currency_symbol precedes the value for a nonnegative formatted monetary quantity; 0 if it does not.	UCHAR_MAX
char p_sep_by_space	1 if the currency_symbol is separated by a space from the value of a nonnegative formatted monetary quantity; 0 if it does not.	UCHAR_MAX
char n_cs_precedes	1 if the currency_symbol precedes the value for a negative formatted monetary quantity; 0 if it does not.	UCHAR_MAX
char n_sep_by_space	1 if the currency_symbol is separated by a space from the value of a negative formatted monetary quantity; 0 if it does not.	UCHAR_MAX
char p_sign_posn	Value indicating the position of the positive_sign for a nonnegative formatted monetary quantity.	UCHAR_MAX
char n_sign_posn	Value indicating the position of the negative_sign for a negative formatted monetary quantity.	UCHAR_MAX

Pointers to strings with a value of "" indicate that the value is not available in the C locale or is of zero length. Elements with char types with a value of UCHAR_MAX indicate that the value is not available in the current locale.

The n_sign_posn and p_sign_posn elements can have the following values:

Value Meaning

- 0 The quantity and currency_symbol are enclosed in parentheses.
- 1 The sign precedes the quantity and currency_symbol.
- The sign follows the quantity and currency_symbol. 2
- 3 The sign precedes the currency_symbol.
- The sign follows the currency_symbol.

Grouping Example

Table 1. Grouping Example

Locale Source	Grouping String	Number	Formatted Number
-1	0x00	123456789	123456789
3	0x0300	123456789	123,456,789
3;-1	0x03FF00	123456789	123456,789
3;2;1	0x03020100	123456789	1,2,3,4,56,789

Monetary Formatting Example:

1

Table 2. Monetary Formatting Example

Country	Positive Format	Negative Format	International Format
Italy	L.1.230	-L.1.230	ITL.1.230
Netherlands	F 1.234,56	F -1.234,56	NLG 1.234,56
Norway	kr1.234,56	kr1.234,56-	NOK1.234,56
Switzerland	SFRs.1,234.56	SFrx.1,234.56C	CHF 1,234.56

The above table was generated by locales withe the following monetary fields:

Table 3. Monetary Fields

	Italy	Netherlands	Norway	Switzerland
int_curr_symbol	"ITL."	"NLG"	"NOK"	"CHF"
currency_symbol	"L."	"F"	"kr"	"SFrs."
mon_decimal_point	""	" "	"",	"."
mon_thousands_sep	"."	"."	"."	"",
mon_grouping	"\3"	"\3"	"\3"	"\3"
positive_sign	""	""	""	""
negative_sign	"_"	"_"	"_"	"C"
int_frac_digits	0	2	2	2
frac_digits	0	2	2	2
p_cs_precedes	1	1	1	1
p_sep_by_space	0	1	0	0
n_cs_preceds	1	1	1	1
n_sep_by_space	0	1	0	0
p_sep_posn	1	1	1	1
n_sign_posn	1	4	2	2

Return Value

The localeconv() function returns a pointer to the structure.

Example that uses *CLD locale objects

This example prints out the default decimal point for your locale and then the decimal point for the LC_C_FRANCE locale.

```
#include <stdio.h>
#include <locale.h>
int main(void) {
  char * string;
  struct lconv * mylocale;
  mylocale = localeconv();
  /* Display default decimal point */
  printf("Default decimal point is a %s\n", mylocale->decimal point);
  if (NULL != (string = setlocale(LC_ALL, LC_C_FRANCE))) {
      mylocale = localeconv();
     /* A comma is set to be the decimal point when the locale is LC C FRANCE*/
     printf("France's decimal point is a %s\n", mylocale->decimal point);
  } else {
     printf("setlocale(LC ALL, LC C FRANCE) returned <NULL>\n");
  return 0;
Example that uses *LOCALE objects
 This example prints out the default decimal point for
the C locale and then the decimal point for the French
locale using a *LOCALE object called
"QSYS.LIB/MYLIB.LIB/LC FRANCE.LOCALE".
 Step 1: Create a French *LOCALE object by entering the command
   CRTLOCALE LOCALE('QSYS.LIB/MYLIB.LIB/LC_FRANCE.LOCALE') +
             SRCFILE('QSYS.LIB/QSYSLOCALE.LIB/QLOCALESRC.FILE/ +
                     FR FR.MBR') CCSID(297)
 Step 2: Compile the following C source, specifying
         LOCALETYPE(*LOCALE) on the compilation command.
 Step 3: Run the program.
 ***********************
#include <stdio.h>
#include <locale.h>
int main(void) {
  char * string;
  struct lconv * mylocale;
  mylocale = localeconv();
  /* Display default decimal point */
  printf("Default decimal point is a %s\n", mylocale->decimal point);
  if (NULL != (string = setlocale(LC_ALL,
      "QSYS.LIB/MYLIB.LIB/LC FRANCE.LOCALE"))) {
     mylocale = localeconv();
     /* A comma is set to be the decimal point in the French locale */
     printf("France's decimal point is a %s\n", mylocale->decimal_point);
  } else {
     printf("setlocale(LC ALL, \"QSYS.LIB/MYLIB.LIB/LC FRANCE.LOCALE\") \
             returned <NULL>\n");
  return 0;
```

- "setlocale() Set Locale" on page 308
- "<locale.h>" on page 7

localtime() — Convert Time

Format

```
#include <time.h>
struct tm *localtime(const time_t *timeval);
```

Language Level: ANSI

Thread Safe: NO. Use localtime_r() instead.

Description

The localtime() function converts a time value, in seconds, to a structure of type *tm*.

Note: This function is locale sensitive.

The localtime() function breaks down *timeval*, corrects for the local time zone and Daylight Saving Time, if appropriate, and stores the corrected time in a structure of type *tm*.

The time value is usually obtained by a call to the time() function.

Note:

- The gmtime() and localtime() functions may use a common, statically allocated buffer for the conversion. Each call to one of these functions may destroy the result of the previous call. The ctime_r(), gmtime_r(), and localtime_r() functions do not use a common, statically-allocated buffer. These functions can be used in the place of the asctime(), ctime(), gmtime() and localtime() functions if reentrancy is desired.
- Calendar time is the number of seconds that have elapsed since EPOCH, which is 00:00:00, January 1, 1970 Universal Coordinate Time (UTC).
- On the iSeries, the time() function uses the QUTCOFFSET system value for calculating UTC. The QUTCOFFSET value specifies the difference in hours and minutes between UTC, also known as Greenwich mean time, and the current system time. You can override the QUTCOFFSET value by specifying the TZDIFF and TZNAME category of the current locale.

Return Value

The localtime() function returns a pointer to the structure result. There is no error return value.

Example that uses localtime()

This example queries the system clock and displays the local time.

```
#include <time.h>
#include <stdio.h>
int main(void)
  struct tm *newtime;
  time t ltime;
  time(localtime());
  newtime = localtime(localtime());
  printf("The date and time is %s", asctime(newtime));
/******* If the local time is 3:00 p.m. May 31, 1993, *******
******************* output should be: ***********
The date and time is Mon May 31 15:00:00 1993
```

- "asctime() Convert Time to Character String" on page 40
- "asctime_r() Convert Time to Character String (Restartable)" on page 42
- "ctime() Convert Time to Character String" on page 66
- "ctime_r() Convert Time to Character String (Restartable)" on page 67
- "gmtime() Convert Time" on page 141
- "gmtime_r() Convert Time (Restartable)" on page 143
- "mktime() Convert Local Time" on page 193
- "localtime_r() Convert Time (Restartable)"
- "setlocale() Set Locale" on page 308
- "time() Determine Current Time" on page 373
- "<time.h>" on page 17

localtime_r() — Convert Time (Restartable)

Format

```
#include <time.h>
struct tm *localtime r(const time t *timeval, struct tm *result);
```

Description

This function is the restartable version of localtime().

The local time r() function converts a time value, in seconds, to a structure of type tm.

The local time r() function breaks down timeval, corrects for the local time zone and Daylight Saving Time, if appropriate, and stores the corrected time in a structure of type tm.

The time value is usually obtained by a call to the time() function.

Notes:

- 1. Calendar time is the number of seconds that have elapsed since EPOCH, which is 00:00:00, January 1, 1970 Universal Coordinate Time (UTC).
- 2. On the iSeries, the time() function uses the QUTCOFFSET system value for calculating UTC. The QUTCOFFSET value specifies the difference in hours and

minutes between UTC, also known as Greenwich mean time, and the current system time. You can override the QUTCOFFSET value by specifying the TZDIFF and TZNAME category of the current locale.

Return Value

The localtime_r() returns a pointer to the structure result. There is no error return value.

Example that uses localtime r()

This example queries the system clock and displays the local time.

```
#include <time.h>
#include <stdio.h>
int main(void)
  struct tm newtime;
  time t ltime;
  char buf[50];
   time(localtime());
  localtime r(localtime(), &newtime);
  printf("The date and time is %s", asctime_r(&newtime, buf));
/******** If the local time is 3:00 p.m. May 31, 1993, *******
******************** output should be: ***********
The date and time is Mon May 31 15:00:00 1993
*/
```

Related Information

- "asctime() Convert Time to Character String" on page 40
- "asctime_r() Convert Time to Character String (Restartable)" on page 42
- "ctime() Convert Time to Character String" on page 66
- "ctime_r() Convert Time to Character String (Restartable)" on page 67
- "gmtime() Convert Time" on page 141
- "gmtime_r() Convert Time (Restartable)" on page 143
- "localtime() Convert Time" on page 163
- "mktime() Convert Local Time" on page 193
- "time() Determine Current Time" on page 373
- "<time.h>" on page 17

log() — Calculate Natural Logarithm

Format

```
#include <math.h>
double log(double x);
```

Language Level: ANSI

Description

The log() function calculates the natural logarithm (base e) of x.

Return Value

The log() function returns the computed value. If x is negative, log() sets errno to EDOM and may return the value -HUGE_VAL. If x is zero, log() returns the value -HUGE_VAL, and may set errno to ERANGE.

Example that uses log()

This example calculates the natural logarithm of 1000.0.

Related Information

- "exp() Calculate Exponential Function" on page 73
- "log10() Calculate Base 10 Logarithm"
- "pow() Compute Power" on page 199
- "<math.h>" on page 8

log10() — Calculate Base 10 Logarithm

Format

```
#include <math.h>
double log10(double x);
```

Language Level: ANSI

Description

The log10() function calculates the base 10 logarithm of x.

Return Value

The log10() function returns the computed value. If x is negative, log10()sets errno to EDOM and may return the value -HUGE_VAL. If x is zero, the log10() function returns the value -HUGE_VAL, and may set errno to ERANGE.

Example that uses log10()

This example calculates the base 10 logarithm of 1000.0.

```
#include <math.h>
#include <stdio.h>
int main(void)
  double x = 1000.0, y;
  y = log10(x);
  printf("The base 10 logarithm of %lf is %lf\n", x, y);
/******* Output should be similar to: ********
The base 10 logarithm of 1000.000000 is 3.000000
```

- "exp() Calculate Exponential Function" on page 73
- "log() Calculate Natural Logarithm" on page 165
- "pow() Compute Power" on page 199
- "<math.h>" on page 8

_Itoa - Convert Long Integer to String

Format

```
#include <stdlib.h>
char * ltoa(long value, char *string, int radix);
```

Note: The _ltoa function is supported only for C++, not for C.

Language Level: Extension

Description

_ltoa converts the digits of the given long integer value to a character string that ends with a null character and stores the result in *string*. The *radix* argument specifies the base of value; it must be in the range 2 to 36. If radix equals 10 and value is negative, the first character of the stored string is the minus sign (-).

Note: The space allocated for *string* must be large enough to hold the returned string. The function can return up to 33 bytes including the null character $(\0).$

Return Value

Itoa returns a pointer to *string*. There is no error return value.

Example that uses _ltoa()

This example converts the integer value -255L to a decimal, a binary, and a hex value, and stores its character representation in the array buffer.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
  char buffer[35];
  char *p;
```

```
p = _ltoa(-255L, buffer, 10);
printf("The result of _ltoa(-255) with radix of 10 is s\n'', p);
p = itoa(-255L, buffer, 2);
printf("The result of _ltoa(-255) with radix of 2\n
                                                             is %s\n", p);
p = _itoa(-255L, buffer, 16);
printf("The result of ltoa(-255) with radix of 16 is %s\n", p);
return 0;
```

The output should be:

```
The result of Itoa(-255) with radix of 10 is -255
The result of ltoa(-255) with radix of 2
    is 111111111111111111111111100000001
The result of _ltoa(-255) with radix of 16 is ffffff01
```

Related Information:

- atol
- "_gcvt Convert Floating-Point to String" on page 132
- "_itoa Convert Integer to String" on page 154
- "_ultoa Convert Unsigned Long Integer to String" on page 380
- wcstol
- <stdlib.h>

longimp() — Restore Stack Environment

Format

```
#include <setimp.h>
void longjmp(jmp buf env, int value);
```

Language Level: ANSI

Description

The longjmp() function restores a stack environment previously saved in *env* by the setjmp() function. The setjmp() and longjmp() functions provide a way to perform a non-local goto. They are often used in signal handlers.

A call to the setjmp() function causes the current stack environment to be saved in env. A subsequent call to longjmp() restores the saved environment and returns control to a point in the program corresponding to the setjmp()call. Processing resumes as if the setjmp() call had just returned the given value.

All variables (except register variables) that are accessible to the function that receives control contain the values they had when longjmp() was called. The values of register variables are unpredictable. Nonvolatile auto variables that are changed between calls to the setjmp() and longjmp() functions are also unpredictable.

Note: Ensure that the function that calls the setjmp() function does not return before you call the corresponding longjmp() function. Calling longjmp() after the function calling the setjmp() function returns causes unpredictable program behavior.

The value argument must be nonzero. If you give a zero argument for value, longjmp() substitutes 1 in its place.

Return Value

The longjmp() function does not use the normal function call and return mechanisms; it has no return value.

Example that uses longjmp()

This example saves the stack environment at the statement:

```
if(setjmp(mark) != 0) ...
```

When the system first performs the if statement, it saves the environment in mark and sets the condition to FALSE because the setjmp() function returns a 0 when it saves the environment. The program prints the message:

```
setjmp has been called
```

The subsequent call to function p() tests for a local error condition, which can cause it to call the longjmp() function. Then, control returns to the original setjmp() function using the environment saved in mark. This time, the condition is TRUE because -1 is the return value from the longjmp() function. The example then performs the statements in the block, prints longjmp() has been called, calls the recover() function, and leaves the program.

```
#include <stdio.h>
#include <setjmp.h>
#include <stdlib.h>
jmp buf mark;
void p(void);
void recover(void);
int main(void)
  if (setjmp(mark) != 0)
    printf("longjmp has been called\n");
    recover();
    exit(1);
  printf("setjmp has been called\n");
  printf("Calling function p()\n");
  p();
  printf("This point should never be reached\n");
void p(void)
  printf("Calling longjmp() from inside function p()\n");
  longjmp(mark, -1);
  printf("This point should never be reached\n");
void recover(void)
  printf("Performing function recover()\n");
setjmp has been called
Calling function p()
Calling longjmp() from inside function p()
longjmp has been called
Performing function recover()
```

- "setjmp() Preserve Environment" on page 307
- "<setjmp.h>" on page 13

malloc() — Reserve Storage Block

Format

#include <stdlib.h> void *malloc(size_t size);

Language Level: ANSI

Thread Safe: YES

Description

The malloc() function reserves a block of storage of *size* bytes. Unlike the calloc() function, malloc() does not initialize all elements to 0.

Notes:

- 1. To use Teraspace storage instead of heap storage without changing the C source code, specify the TERASPACE(*YES *TSIFC) parameter on the CRTCMOD compiler command. This maps the malloc() library function to _C_TS_malloc(), its Teraspace storage counterpart. The maximum amount of Teraspace storage that can be allocated by each call to _C_TS_malloc() is 2GB - 224, or 2147483424 bytes. If more than 2147483408 bytes are needed on a single request, call _C_TS_malloc64(unsigned long long int);.
 - For more information, see the *ILE Concepts* manual.
- 2. For current statistics on the teraspace storage being used by MI programs in an activation group, call the _C_TS_malloc_info function. This function returns information including total bytes, allocated bytes and blocks, unallocated bytes and blocks, requested bytes, pad bytes, and overhead bytes. To get more detailed information about the memory structures used by the C TS malloc() and _C_TS_malloc64() functions, call the _C_TS_malloc_debug function. You can use the information this function returns to identify memory corruption problems.

Return Value

The malloc() function returns a pointer to the reserved space. The storage space to which the return value points is suitably aligned for storage of any type of object. The return value is NULL if not enough storage is available, or if size was specified as zero.

Example that uses malloc()

This example prompts for the number of array entries you want and then reserves enough space in storage for the entries. If malloc() was successful, the example assigns values to the entries and prints out each entry; otherwise, it prints out an error.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
 long * array;
               /* start of the array */
 long * index; /* index variable */
 int i;
              /* index variable */
               /* number of entries of the array */
 int num;
 printf( "Enter the size of the array\n" );
 scanf( "%i", &num );
 /* allocate num entries */
 if ( (index = array = (long *) malloc( num * sizeof( long ))) != NULL )
   for ( i = 0; i < num; ++i )
                                                            */
                                      /* put values in array
                                     /* using pointer notation */
      *index++ = i;
   for ( i = 0; i < num; ++i )
                                     /* print the array out */
     printf( "array[ %i ] = %i\n", i, array[i] );
 else { /* malloc error */
   perror( "Out of storage" );
   abort();
/******* Output should be similar to: ********
Enter the size of the array
array[0] = 0
array[1] = 1
array[2] = 2
array[3] = 3
array[4] = 4
```

- "calloc() Reserve and Initialize Storage" on page 56
- "free() Release Storage Blocks" on page 109
- "realloc() Change Reserved Storage Block Size" on page 234
- "<stdlib.h>" on page 16

mblen() — Determine Length of a Multibyte Character

Format

```
#include <stdlib.h>
int mblen(const char *string, size_t n);
```

Language Level: ANSI

Thread Safe: NO. Use mbrlen() instead.

Description

The mblen() function determines the length in bytes of the multibyte character pointed to by *string*. A maximum of n bytes is examined.

Return Value

If *string* is NULL, the mblen() function returns:

- Non-zero if the active locale allows mixed-byte strings. The function initializes the state variable.
- · Zero otherwise.

If *string* is not NULL, mblen() returns:

- Zero if *string* points to the null character.
- The number of bytes comprising the multibyte character.
- -1 if *string* does not point to a valid multibyte character.

Note: The mblen(), mbtowc(), and wctomb() functions use their own statically allocated storage and are therefore not restartable. However, mbrlen(), mbrtowc(), and wcrtomb() are restartable.

Example that uses mblen()

This example uses mblen() and mbtowc() to convert a multibyte character into a single wide character.

```
#include <stdio.h>
#include <stdlib.h>
int length, temp;
char string [6] = "w":
wchar t arr[6];
int main(void)
   /* Initialize internal state variable */
  length = mblen(NULL, MB CUR MAX);
   /* Set string to point to a multibyte character */
  length = mblen(string, MB CUR MAX);
  temp = mbtowc(arr,string,length);
  printf("wide character string: arr[1] = L'\0';",arr);
```

Related Information

- "mbrlen() Determine Length of a Multibyte Character (Restartable)"
- "mbtowc() Convert Multibyte Character to a Wide Character" on page 186
- "mbstowcs() Convert a Multibyte String to a Wide Character String" on page 182
- "strlen() Determine String Length" on page 341
- "wcslen() Calculate Length of Wide-Character String" on page 409
- "wctomb() Convert Wide Character to Multibyte Character" on page 436
- "<stdlib.h>" on page 16

mbrlen() — Determine Length of a Multibyte Character (Restartable)

Format

```
#include <wchar.h>
size_t mbrlen (const char *s, size_t n, mbstate_t *ps);
```

Language Level: ANSI

Description

This function is the restartable version of mblen().

The mbrlen() function determines the length of a multibyte character.

n is the number of bytes (at most) of the multibyte string to examine.

This function differs from its corresponding internal-state multibyte character function in that it has an extra parameter, ps of type pointer to mbstate_t that points to an object that can completely describe the current conversion state of the associated multibyte character sequence. If ps is a NULL pointer, mbrlen() behaves like mblen().

mbrlen() is a restartable version of mblen(). In other words, shift-state information is passed as one of the arguments (ps represents the initial shift) and is updated on exit. With mbrlen(), you can switch from one multibyte string to another, provided that you have kept the shift-state information.

This function is affected by the LC_CTYPE of the current locale.

Note: This function is only accessible when you specify LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) on the compilation command.

Return Value

If s is a null pointer, if the active locale allows mixed-byte strings, and if the active locale initializes the state variabe, the mbrlen() function returns nonzero. If the active locale does not allow mixed byte strings, zero will be returned.

If *s* is not a null pointer, the mbrlen() function returns one of the following:

If *s* is a NULL string (*s* points to the NULL character).

positive

If the next n or fewer bytes comprise a valid multibyte character. The value returned is the number of bytes that comprise the multibyte character.

$(size_t)-1$

If s does not point to a valid multibyte character.

$(size_t)-2$

If the next *n* or fewer bytes contribute to an incomplete but potentially valid character and all *n* bytes have been processed

Example that uses mbrlen()

```
/* This program is compiled with LOCALETYPE(*LOCALE) and
/* SYSIFCOPT(*IFSIO)
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <wchar.h>
#include <errno.h>
#define LOCNAME
                    "qsys.lib/JA JP.locale"
#define LOCNAME_EN "qsys.lib/EN_US.locale"
int main(void)
```

1

```
int length, sl = 0;
 char string[10];
 mbstate t ps = 0;
 memset(\overline{\text{string}}, '\0', 10);
 string[0] = 0xC1;
 string[1] = 0x0E;
 string[2] = 0x41;
 string[3] = 0x71;
 string[4] = 0x41;
 string[5] = 0x72;
 string[6] = 0x0F;
 string[7] = 0xC2;
 /* In this first example we will find the length of
 /* of a multibyte character when the CCSID of locale
 /* associated with LC CTYPE is 37.
 /* For single byte cases the state will always
 /* remain in the initial state 0
 if (setlocale(LC_ALL, LOCNAME_EN) == NULL)
     printf("setlocale failed.\n");
 length = mbrlen(string, MB CUR MAX, &ps);
 /* In this case length is 1, which is always the case for */
 /* single byte CCSID */
 printf("length = %d, state = %d\n\n", length, ps);
 printf("MB_CUR_MAX: %d\n\n", MB_CUR_MAX);
 /* Now lets try a multibyte example. We first must set the */
 /* locale to a multibyte locale. We choose a locale with
 /* CCSID 5026 */
 if (setlocale(LC ALL, LOCNAME) == NULL)
     printf("setlocale failed.\n");
 length = mbrlen(string, MB_CUR_MAX, &ps);
 /* The first is single byte so length is 1 and
 /* the state is still the initial state 0
 printf("length = %d, state = %d\n\n", length, ps);
 printf("MB CUR MAX: %d\n\n", MB CUR MAX);
 sl += length;
 length = mbrlen(&string[s1], MB CUR MAX, &ps);
 /* The next character is a mixed byte. Length is 3 to
 /* account for the shiftout 0x0e. State is
 /* changed to double byte state.
 printf("length = %d, state = %d\n\n", length, ps);
 sl += length;
length = mbrlen(&string[s1], MB CUR MAX, &ps);
/* The next character is also a double byte character.
/* The state is changed to initital state since this was
/* the last double byte character. Length is 3 to
/* account for the ending 0x0f shiftin.
printf("length = %d, state = %d\n\n", length, ps);
sl += length;
```

```
length = mbrlen(&string[s1], MB CUR MAX, &ps);
   /* The next character is single byte so length is 1 and
   /* state remains in initial state.
   printf("length = %d, state = %d\n\n", length, ps);
/* The output should look like this:
length = 1, state = 0
MB CUR MAX: 1
length = 1, state = 0
MB_CUR_MAX: 4
length = 3, state = 2
length = 3, state = 0
length = 1, state = 0
                                   */
* * * End of File * * *
```

- "mblen() Determine Length of a Multibyte Character" on page 172
- "mbtowc() Convert Multibyte Character to a Wide Character" on page 186
- "mbrtowc() Convert a Multibyte Character to a Wide Character (Restartable)"
- "mbsrtowcs() Convert a Multibyte String to a Wide Character String (Restartable)" on page 180
- "setlocale() Set Locale" on page 308
- "wcrtomb() Convert a Wide Character to a Multibyte Character (Restartable)" on page 396
- "wcsrtombs() Convert Wide Character String to Multibyte String (Restartable)" on page 418
- "<locale.h>" on page 7
- "<wchar.h>" on page 18

mbrtowc() — Convert a Multibyte Character to a Wide Character (Restartable)

Format

```
#include <wchar.h>
size t mbrtowc (wchar t *pwc, const char *s, size t n, mbstate t *ps);
```

Description

This function is the restartable version of the mbtowc() function.

If s is a null pointer, the mbrtowc() function determines the number of bytes necessary to enter the initial shift state (zero if encodings are not state-dependent or if the initial conversion state is described). In this situation, the value of the pwc parameter will be ignored and the resulting shift state described will be the initial conversion state.

If s is not a null pointer, the mbrtowc() function determines the number of bytes that are in the multibyte character (and any leading shift sequences) pointed to by s, produces the value of the corresponding multibyte character and if pwc is not a null pointer, stores that value in the object pointed to by pwc. If the corresponding multibyte character is the null wide character, the resulting state will be reset to the initial conversion state.

This function differs from its corresponding internal-state multibyte character function in that it has an extra parameter, ps of type pointer to mbstate_t that points to an object that can completely describe the current conversion state of the associated multibyte character sequence. If ps is NULL, this function uses an internal static variable for the state.

At most, *n* bytes of the multibyte string are examined.

Note: If the program is compiled with LOCALETYPE(*LOCALEUCS2), this function is accessible, but UNICODE is not supported at this time. It will behave as if it were compiled with LOCALETYPE(*LOCALE).

This function is accessible only if LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

Return Value

If s is a null pointer, the mbrtowc() function returns the number of bytes necessary to enter the initial shift state. The value returned must be less than the MB_CUR_MAX macro.

If *s* is not a null pointer, the mbrtowc() function returns one of the following:

If the next n or fewer bytes form the multibyte character that corresponds to the null wide character.

positive

If the next n or fewer bytes form a valid multibyte character. The value returned is the number of bytes that constitute the multibyte character.

$(size_t)-2$

If the next *n* bytes form an incomplete (but potentially valid) multibyte character, and all n bytes have been processed. It is unspecified whether this can occur when the value of n is less than the value of the MB_CUR_MAX macro.

$(size_t)-1$

If an encoding error occurs (when the next *n* or fewer bytes do not form a complete and correct multibyte character). The value of the macro EILSEQ is stored in errno, but the conversion state is unchanged.

Note: When a -2 value is returned, the string would contain redundant shift-out and shift-in characters. To continue processing the multibyte string, increment the pointer by the value n, and call mbrtowc() again.

Example that uses mbrtowc()

```
/* This program is compiled with LOCALETYPE(*LOCALE) and
                                                          */
/* SYSIFCOPT(*IFSIO)
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
```

```
#include <wchar.h>
#include <errno.h>
#define LOCNAME
                     "/qsys.lib/JA_JP.locale"
#define LOCNAME EN "/qsys.lib/EN US.locale"
int main(void)
   int length, sl = 0;
   char string[10];
   wchar t buffer[10];
   mbstate_t ps = 0;
memset(string, '\0', 10);
   string[0] = 0xC1;
    string[1] = 0x0E;
   string[2] = 0x41;
    string[3] = 0x71;
    string[4] = 0x41;
    string[5] = 0x72;
    string[6] = 0x0F;
    string[7] = 0xC2;
   /* In this first example we will convert
    /* a multibyte character when the CCSID of locale
   /* associated with LC CTYPE is 37.
    /* For single byte cases the state will always
    /* remain in the initial state 0
    if (setlocale(LC ALL, LOCNAME EN) == NULL)
        printf("setlocale failed.\n");
    length = mbrtowc(buffer, string, MB CUR MAX, &ps);
    /* In this case length is 1, and C1 is converted 0x00C1
   printf("length = %d, state = %d\n\n", length, ps);
    printf("MB_CUR_MAX: %d\n\n", MB_CUR_MAX);
    /* Now lets try a multibyte example. We first must set the */
    /* locale to a multibyte locale. We choose a locale with
    /* CCSID 5026 */
    if (setlocale(LC ALL, LOCNAME) == NULL)
        printf("setlocale failed.\n");
    length = mbrtowc(buffer, string, MB CUR MAX, &ps);
    /* The first is single byte so length is 1 and
    /* the state is still the initial state 0. C1 is converted*/
    /* to 0x00C1
                     */
   printf("length = %d, state = %d\n\n", length, ps);
   printf("MB_CUR_MAX: %d\n\n", MB_CUR_MAX);
   sl += length;
    length = mbrtowc(&buffer[1], &string[s1], MB CUR MAX, &ps);
    /* The next character is a mixed byte. Length is 3 to
                                                                */
    /* account for the shiftout 0x0e. State is
    /* changed to double byte state. 0x4171 is copied into
    /* the buffer
   printf("length = %d, state = %d\n\n", length, ps);
    sl += length;
```

```
length = mbrtowc(&buffer[2], &string[s1], MB CUR MAX, &ps);
    /* The next character is also a double byte character.
    /* The state is changed to initial state since this was
    /* the last double byte character. Length is 3 to
    /* account for the ending 0x0f shiftin. 0x4172 is copied */
    /* into the buffer. */
    printf("length = %d, state = %d\n\n", length, ps);
    sl += length;
    length = mbrtowc(&buffer[3], &string[s1], MB CUR MAX, &ps);
    /* The next character is single byte so length is 1 and
    /* state remains in initial state. 0xC2 is converted to
               The buffer now has the value:
    /* 0x00C2.
                                                               */
    /* 0x00C14171417200C2
    printf("length = %d, state = %d\n\n", length, ps);
/* The output should look like this:
length = 1, state = 0
MB_CUR_MAX: 1
length = 1, state = 0
MB CUR MAX: 4
length = 3, state = 2
length = 3, state = 0
length = 1, state = 0
                                   */
```

- "mblen() Determine Length of a Multibyte Character" on page 172
- "mbrlen() Determine Length of a Multibyte Character (Restartable)" on page 173
- "mbsrtowcs() Convert a Multibyte String to a Wide Character String (Restartable)" on page 180
- "setlocale() Set Locale" on page 308
- "wcrtomb() Convert a Wide Character to a Multibyte Character (Restartable)" on page 396
- "wcsrtombs() Convert Wide Character String to Multibyte String (Restartable)" on page 418
- "<locale.h>" on page 7
- "<wchar.h>" on page 18

mbsinit() — Test State Object for Initial State

Format

```
#include <wchar.h>
int mbsinit (const mbstate t *ps);
```

Description

If ps is not a null pointer, the mbsinit() function specifies whether the pointed to mbstate_t object describes an initial conversion state.

Note: This function is only accessible if LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

Return Value

The mbsinit() function returns nonzero if ps is a null pointer or if the pointed to object describes an initial conversion state. Otherwise, it returns zero.

Example that uses mbsinit()

This example checks the conversion state to see if it is the initial state.

```
#include <stdio.h>
#include <wchar.h>
#include <stdlib.h>
main()
           *string = "ABC";
   char
  mbstate_t state = 0;
   wchar_t wc;
   int
        rc;
   rc = mbrtowc(&wc, string, MB_CUR_MAX, &state);
   if (mbsinit(&state))
     printf("In initial conversion state\n");
```

Related Information

- "mbrlen() Determine Length of a Multibyte Character (Restartable)" on page 173
- "mbrtowc() Convert a Multibyte Character to a Wide Character (Restartable)" on page 176
- "mbsrtowcs() Convert a Multibyte String to a Wide Character String (Restartable)"
- "setlocale() Set Locale" on page 308
- "wcrtomb() Convert a Wide Character to a Multibyte Character (Restartable)" on page 396
- "wcsrtombs() Convert Wide Character String to Multibyte String (Restartable)" on page 418
- "<locale.h>" on page 7
- "<wchar.h>" on page 18

mbsrtowcs() — Convert a Multibyte String to a Wide Character String (Restartable)

Format

```
#include <wchar.h>
size t mbsrtowcs (wchar t *dst, const char **src, size t len,
                  mbstate t *ps);
```

Thread Safe: YES, if ps is not NULL.

1

Description

This function is the restartable version of mbstowcs().

The mbsrtowcs() function converts a sequence of multibyte characters that begins in the conversion state described by ps from the array indirectly pointed to by src into a sequence of corresponding wide characters. It then stores the converted characters into the array pointed to by dst.

Conversion continues up to and including an ending null character, but the ending null wide character will not be stored. Conversion will stop earlier in two cases: when a sequence of bytes are reached that do not form a valid multibyte character, or (if *dst* is not a null pointer) when *len* wide characters have been stored into the array pointed to by *dst*. Each conversion takes place as if by a call to mbrtowc() function.

If *dst* is not a null pointer, the pointer object pointed to by *src* will be assigned either a null pointer (if conversion stopped due to reaching an ending null character) or the address just past the last multibyte character converted. If conversion stopped due to reaching an ending null character, the initial conversion state is described.

The behavior of mbsrtowcs() is affected by the LC_CTYPE category of the current locale.

Note: This function is accessible only if LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

If the program is compiled LOCALETYPE(*LOCALEUCS2), this function will behave the same as if it were compiled with LOCALETYPE(*LOCALE), since UCS2 is not supported by this function.

Return Value

If the input string does not begin with a valid multibyte character, an encoding error occurs, the mbsrtowcs() function stores the value of the macro EILSEQ in errno, and returns (size_t) -1, but the conversion state will be unchanged. Otherwise, it returns the number of multibyte characters successfully converted, which is the same as the number of array elements modified when *dst* is not a null pointer.

Example that uses mbsrtowcs()

```
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <locale.h>
#define SIZE 10
int main(void)
            mbs1[] = "abc";
  char
              mbs2[] = "\x81\x41" "m" "\x81\x42";
  char
  const char *pmbs1 = mbs1;
  const char *pmbs2 = mbs2;
  mbstate t ss1 = 0;
  mbstate_t ss2 = 0;
  wchar t
            wcs1[SIZE], wcs2[SIZE];
  if (NULL == setlocale(LC ALL, "/qsys.lib/locale.lib/ja jp939.locale"))
     printf("setlocale failed.\n");
     exit(EXIT FAILURE);
  mbsrtowcs(wcs1, &pmbs1, SIZE, 5722-SS1);
  mbsrtowcs(wcs2, &pmbs2, SIZE, &ss2);
  printf("The first wide character string is %ls.\n", wcs1);
  printf("The second wide character string is %ls.\n", wcs2);
  return 0:
  /***************
     The output should be similar to:
     The first wide character string is abc.
     The second wide character string is Am B.&#26
```

Also, see the examples for "mbrtowc() — Convert a Multibyte Character to a Wide Character (Restartable)" on page 176.

Related Information

- "mblen() Determine Length of a Multibyte Character" on page 172
- "mbrlen() Determine Length of a Multibyte Character (Restartable)" on page 173
- "mbrtowc() Convert a Multibyte Character to a Wide Character (Restartable)" on page 176
- "mbstowcs() Convert a Multibyte String to a Wide Character String"
- "setlocale() Set Locale" on page 308
- "wcrtomb() Convert a Wide Character to a Multibyte Character (Restartable)" on page 396
- "wcsrtombs() Convert Wide Character String to Multibyte String (Restartable)" on page 418
- "<locale.h>" on page 7
- "<wchar.h>" on page 18

mbstowcs() — Convert a Multibyte String to a Wide Character String

Format

```
#include <stdlib.h>
size t mbstowcs(wchar t *pwc, const char *string, size t n);
```

1 Language Level: ANSI

Thread Safe: YES.

Description

The mbstowcs() function determines the length of the sequence of the multibyte characters pointed to by string. It then converts the multibyte character string that begins in the initial shift state into a wide character string, and stores the wide characters into the buffer that is pointed to by pwc. A maximum of n wide characters are written.

The behavior of this function is affected by the LC_CTYPE category of the current locale. If the program is compiled with LOCALETYPE(*LOCALE), and the CCSID that is associated with locale is single-byte, all the multibyte characters are assumed to be single-byte, and are converted to wide characters like this: 0xC1 is converted to 0x00C1. If CCSID of the locale is a multibyte CCSID, then if the multibyte character is a single-byte character, it is converted as is above. If the multibyte character is a double-byte character, (characters between shiftout 0x0e and shiftin 0x0f), the double-byte characters are copied directly, stripping off the shiftout and shiftin characters. For example, 0x0e41710f is converted to 0x4171.

If the program is compiled LOCALETYPE(*LOCALEUCS2), the multibyte character string is converted from the CCSID of the locale to UNICODE as if iconv() were called.

Note: This function is accessible only if LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

Return Value

The mbstowcs() function returns the number of wide characters generated, not including any ending null wide characters. If a multibyte character that is not valid is encountered, the function returns (size_t)-1.

Examples that use mbstowcs()

```
/* This program is compiled with LOCALETYPE(*LOCALEUCS2) and
/* SYSIFCOPT(*IFSIO)
#include
          <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <wchar.h>
#include <errno.h>
#define LOCNAME
                   "qsys.lib/JA JP.locale"
#define LOCNAME EN "qsys.lib/EN US.locale"
int main(void)
    int length, sl = 0;
    char string[10];
    char string2[] = "ABC";
    wchar t buffer[10];
   memset(string, '\0', 10);
    string[0] = 0xC1;
    string[1] = 0x0E;
    string[2] = 0x41;
    string[3] = 0x71;
    string[4] = 0x41;
    string[5] = 0x72;
    string[6] = 0x0F;
    string[7] = 0xC2;
    /* In this first example we will convert
    /* a multibyte character when the CCSID of locale
    /* associated with LC_CTYPE is 37.
    if (setlocale(LC ALL, LOCNAME EN) == NULL)
        printf("setlocale failed.\n");
    length = mbstowcs(buffer, string2, 10);
    /* In this case length ABC is converted to UNICODE ABC
    /* or 0x004100420043. Length will be 3.
    printf("length = %d\n\n", length);
    /* Now lets try a multibyte example. We first must set the */
    /* locale to a multibyte locale. We choose a locale with
    /* CCSID 5026 */
    if (setlocale(LC ALL, LOCNAME) == NULL)
        printf("setlocale failed.\n");
    length = mbstowcs(buffer, string, 10);
    /* The buffer now has the value:
    /* 0x004103A103A30042
                                length is 4
    printf("length = %d\n\n", length);
/* The output should look like this:
length = 3
length = 4
                                   */
```

```
/* This program is compiled with LOCALETYPE(*LOCALE) and
/* SYSIFCOPT(*IFSIO)
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <wchar.h>
#include <errno.h>
#define LOCNAME
                     "qsys.lib/JA JP.locale"
#define LOCNAME EN "qsys.lib/EN US.locale"
int main(void)
    int length, sl = 0;
    char string[10];
    char string2[] = "ABC";
   wchar t buffer[10];
   memset(string, '\0', 10);
    string[0] = 0xC1;
    string[1] = 0x0E;
   string[2] = 0x41;
   string[3] = 0x71;
   string[4] = 0x41;
   string[5] = 0x72;
    string[6] = 0x0F;
   string[7] = 0xC2;
    /* In this first example we will convert
    /* a multibyte character when the CCSID of locale
    /* associated with LC_CTYPE is 37.
    if (setlocale(LC ALL, LOCNAME EN) == NULL)
        printf("setlocale failed.\n");
    length = mbstowcs(buffer, string2, 10);
    /* In this case length ABC is converted to
    /* 0x00C100C200C3. Length will be 3.
    printf("length = %d\n\n", length);
    /* Now lets try a multibyte example. We first must set the *
    /* locale to a multibyte locale. We choose a locale with
    /* CCSID 5026 */
    if (setlocale(LC_ALL, LOCNAME) == NULL)
        printf("setlocale failed.\n");
   length = mbstowcs(buffer, string, 10);
    /* The buffer now has the value:
    /* 0x00C14171417200C2
                                 length is 4
    printf("length = %d\n\n", length);
}
/* The output should look like this:
length = 3
length = 4
                                   */
```

• "mblen() — Determine Length of a Multibyte Character" on page 172

- "mbtowc() Convert Multibyte Character to a Wide Character"
- "setlocale() Set Locale" on page 308
- "wcslen() Calculate Length of Wide-Character String" on page 409
- "wcstombs() Convert Wide-Character String to Multibyte String" on page 426
- "<locale.h>" on page 7
- "<stdlib.h>" on page 16
- "<wchar.h>" on page 18

mbtowc() — Convert Multibyte Character to a Wide Character

Format

```
#include <stdlib.h>
int mbtowc(wchar t *pwc, const char *string, size t n);
```

Language Level: ANSI

Thread Safe: NO. Use mbrtowc() instead.

Description

The mbtowc() function first determines the length of the multibyte character pointed to by string. It then converts the multibyte character to a wide character as described in mbstowcs. A maximum of n bytes are examined.

The behavior of this function is affected by the LC_CTYPE category of the current locale.

Return Value

If *string* is NULL, the mbtowc() function returns:

- · Nonzero when the active locale is mixed byte. The function initializes the state variable.
- 0 otherwise.

If *string* is not NULL, the mbtowc() function returns:

- 0 if string points to the null character
- The number of bytes comprising the converted multibyte character
- -1 if *string* does not point to a valid multibyte character.

Example that uses mbtowc()

This example uses the mblen() and mbtowc() functions to convert a multibyte character into a single wide character.

```
#include <stdio.h>
#include <stdib.h>

#define LOCNAME "qsys.lib/mylib.lib/ja_jp959.locale"
/*Locale created from source JA_JP and CCSID 939 */

int length, temp;
char string [] = "\x0e\x41\x71\x0f";
wchar_t arr[6];

int main(void)
{
    /* initialize internal state variable */
    temp = mbtowc(arr, NULL, 0);

    setlocale (LC_ALL, LOCNAME);
    /* Set string to point to a multibyte character. */
    length = mblen(string, MB_CUR_MAX);
    temp = mbtowc(arr,string,length);
    printf("wide character string: %ls",arr);
}
```

- "mblen() Determine Length of a Multibyte Character" on page 172
- "mbstowcs() Convert a Multibyte String to a Wide Character String" on page 182
- "wcslen() Calculate Length of Wide-Character String" on page 409
- "wctomb() Convert Wide Character to Multibyte Character" on page 436
- "<stdlib.h>" on page 16

memchr() — Search Buffer

Format

```
#include <string.h>
void *memchr(const void *buf, int c, size_t count);
```

Language Level: ANSI

Description

The memchr() function searches the first *count* bytes of *buf* for the first occurrence of c converted to an unsigned character. The search continues until it finds c or examines *count* bytes.

Return Value

The memchr() function returns a pointer to the location of c in buf. It returns NULL if c is not within the first c of buf.

Example that uses memchr()

This example finds the first occurrence of "x" in the string that you provide. If it is found, the string that starts with that character is printed.

```
#include <stdio.h>
#include <string.h>
int main(int argc, char ** argv)
 char * result;
 if ( argc != 2 )
   printf( "Usage: %s string\n", argv[0] );
   if ((result = (char *) memchr( argv[1], 'x', strlen(argv[1])) ) != NULL)
     printf( "The string starting with x is s\n", result );
     printf( "The letter x cannot be found in the stringn");
/******* Output should be similar to: ********
The string starting with x is xing
```

- "memcmp() Compare Buffers"
- "memcpy() Copy Bytes" on page 189
- "memmove() Copy Bytes" on page 191
- "wmemchr() —Locate Wide Character in Wide-Character Buffer" on page 442
- "memset() Set Bytes to Value" on page 192
- "strchr() Search for Character" on page 325
- "<string.h>" on page 17

memcmp() — Compare Buffers

Format

```
#include <string.h>
int memcmp(const void *buf1, const void *buf2, size_t count);
```

Language Level: ANSI

Description

The memcmp() function compares the first *count* bytes of *buf1* and *buf2*.

Return Value

The memcmp() function returns a value indicating the relationship between the two buffers as follows:

Value	Meaning
Less than 0	buf1 less than buf2
0	buf1 identical to buf2
Greater than 0	buf1 greater than buf2

Example that uses memcmp()

This example compares first and second arguments passed to main() to determine which, if either, is greater.

```
#include <stdio.h>
#include <string.h>
int main(int argc, char ** argv)
 int len;
 int result;
 if ( argc != 3 )
    printf( "Usage: %s string1 string2\n", argv[0] );
 else
    /* Determine the length to be used for comparison */
    if (strlen( argv[1] ) < strlen( argv[2] ))</pre>
      len = strlen( argv[1] );
      len = strlen( argv[2] );
    result = memcmp( argv[1], argv[2], len );
    printf( "When the first %i characters are compared,\n", len );
    if (result == 0)
      printf( "\"s\" is identical to \"s\"\n", argv[1], argv[2] );
    else if ( result < 0 )
      printf( "\"%s\" is less than \"%s\"\n", argv[1], argv[2] );
    else
      printf( "\"%s\" is greater than \"%s\"\n", argv[1], argv[2] );
}
/******* If the program is passed the arguments *********
******* firststring and secondstring, ********
*****
                     output should be:
                                                    ******
When the first 11 characters are compared,
"firststring" is less than "secondstring"
Related Information
• "memchr() — Search Buffer" on page 187

    "memcpy() — Copy Bytes"

• "wmemcmp() —Compare Wide-Character Buffers" on page 443
```

- "memmove() Copy Bytes" on page 191
- "memset() Set Bytes to Value" on page 192
- "strcmp() Compare Strings" on page 326
- "<string.h>" on page 17

memcpy() — Copy Bytes

Format

```
#include <string.h>
void *memcpy(void *dest, const void *src, size_t count);
```

Language Level: ANSI

Description

The memcpy() function copies count bytes of src to dest. The behavior is undefined if copying takes place between objects that overlap. The memmove() function allows copying between objects that may overlap.

Return Value

The memcpy() function returns a pointer to *dest*.

Example that uses memcpy()

This example copies the contents of source to target.

```
#include <string.h>
#include <stdio.h>
#define MAX LEN 80
char source[ MAX LEN ] = "This is the source string";
char target[ MAX LEN ] = "This is the target string";
int main(void)
 printf( "Before memcpy, target is \"%s\"\n", target );
 memcpy( target, source, sizeof(source));
 printf( "After memcpy, target becomes \"%s\"\n", target );
Before memcpy, target is "This is the target string"
After memcpy, target becomes "This is the source string"
```

Related Information

- "memchr() Search Buffer" on page 187
- "memcmp() Compare Buffers" on page 188
- "wmemcpy() —Copy Wide-Character Buffer" on page 444
- "memmove() Copy Bytes" on page 191
- "memset() Set Bytes to Value" on page 192
- "strcpy() Copy Strings" on page 330
- "<string.h>" on page 17

memicmp - Compare Bytes

Format

```
#include <string.h>
                     // also in <memory.h>
int memicmp(void *buf1, void *buf2, unsigned int cnt);
```

Note: The memicmp function is supported only for C++, not for C.

Language Level: Extension

Description

The memicmp function compares the first cnt bytes of buf1 and buf2 without regard to the case of letters in the two buffers. The function converts all uppercase characters into lowercase and then performs the comparison.

Return Value

The return value of memicmp indicates the result as follows:

Value	Meaning
Less than 0	buf1 less than buf2
0	buf1 identical to buf2
Greater than 0	buf1 greater than buf2

Example that uses memicmp()

This example copies two strings that each contain a substring of 29 characters that are the same except for case. The example then compares the first 29 bytes without regard to case.

```
#include <stdio.h>
#include <string.h>
char first[100],second[100];
int main(void)
    int result;
    strcpy(first, "Those Who Will Not Learn From History");
strcpy(second, "THOSE WHO WILL NOT LEARN FROM their mistakes");
    printf("Comparing the first 29 characters of two strings.\n");
    result = memicmp(first, second, 29);
    printf("The first 29 characters of String 1 are ");
    if (result < 0)
       printf("less than String 2.\n");
    else
        if (0 == result)
           printf("equal to String 2.\n");
           printf("greater than String 2.\n");
    return 0;
}
```

The output should be:

Comparing the first 29 characters of two strings. The first 29 characters of String 1 are equal to String 2

Related Information:

- memchr
- memcmp
- memcpy
- memmove
- memset
- strcmp
- "strcmpi Compare Strings Without Case Sensitivity" on page 328
- "stricmp Compare Strings without Case Sensitivity" on page 340
- "strnicmp Compare Substrings Without Case Sensitivity" on page 347
- <memory.h>
- <string.h>

memmove() — Copy Bytes

Format

```
#include <string.h>
void *memmove(void *dest, const void *src, size t count);
```

Language Level: ANSI

Description

The memmove() function copies *count* bytes of *src* to *dest*. This function allows copying between objects that may overlap as if src is first copied into a temporary array.

Return Value

The memmove() function returns a pointer to *dest*.

Example that uses memmove()

This example copies the word "shiny" from position target + 2 to position target +

```
#include <string.h>
#include <stdio.h>
#define SIZE
char target[SIZE] = "a shiny white sphere";
int main( void )
 char * p = target + 8; /* p points at the starting character
                        of the word we want to replace */
 char * source = target + 2; /* start of "shiny" */
 printf( "Before memmove, target is \"%s\"\n", target );
 memmove( p, source, 5 );
 printf( "After memmove, target becomes \"%s\"\n", target );
/****** Expected output: *************
Before memmove, target is "a shiny white sphere"
After memmove, target becomes "a shiny shiny sphere"
```

Related Information

- "memchr() Search Buffer" on page 187
- "memcmp() Compare Buffers" on page 188
- "wmemmove() Copy Wide-Character Buffer" on page 445
- "memcpy() Copy Bytes" on page 189
- "memset() Set Bytes to Value"
- "strcpy() Copy Strings" on page 330
- "<string.h>" on page 17

memset() — Set Bytes to Value

Format

```
#include <string.h>
void *memset(void *dest, int c, size t count);
```

Language Level: ANSI

Description

The memset () function sets the first *count* bytes of *dest* to the value *c*. The value of *c* is converted to an unsigned character.

Return Value

The memset () function returns a pointer to *dest*.

Example that uses memset()

This example sets 10 bytes of the buffer to A and the next 10 bytes to B.

```
#include <string.h>
#include <stdio.h>
#define BUF SIZE 20
int main(void)
  char buffer[BUF SIZE + 1];
  char *string;
  memset(buffer, 0, sizeof(buffer));
  string = (char *) memset(buffer, 'A', 10);
  printf("\nBuffer contents: %s\n", string);
  memset(buffer+10, 'B', 10);
  printf("\nBuffer contents: %s\n", buffer);
/******* Output should be similar to: ********
Buffer contents: AAAAAAAAA
*/
```

Related Information

- "memchr() Search Buffer" on page 187
- "memcmp() Compare Buffers" on page 188
- "memcpy() Copy Bytes" on page 189
- "memmove() Copy Bytes" on page 191
- "<string.h>" on page 17
- "wmemset() Set Wide Character Buffer to a Value" on page 446

mktime() — Convert Local Time

Format

```
#include <time.h>
time t mktime(struct tm *time);
```

Language Level: ANSI

Thread Safe: NO.

Description

The mktime() function converts local time, stored as a tm structure pointed to by time, into a time_t structure suitable for use with other time functions. The values of some structure elements pointed to by time are not restricted to the ranges shown for gmtime().

The values of tm_wday and tm_yday passed to mktime() are ignored and are assigned their correct values on return.

Note:

- Calendar time is the number of seconds that have elapsed since EPOCH, which is 00:00:00, January 1, 1970 Universal Coordinate Time (UTC).
- On the iSeries server, the time function uses the QUTCOFFSET system value for calculating UTC. The QUTCOFFSET value specifies the difference in hours and minutes between UTC, also known as Greenwich mean time, and the current system time. You can override the QUTCOFFSET value by specifying the TZDIFF and TZNAME category of the current locale.

Return Value

The mktime() function returns the calendar time having type time_t. The value (time t)(-1) is returned if the calendar time cannot be represented.

Example that uses mktime()

This example prints the day of the week that is 40 days and 16 hours from the current date.

```
#include <stdio.h>
#include <time.h>
char *wday[] = { "Sunday", "Monday", "Tuesday", "Wednesday",
                "Thursday", "Friday", "Saturday" };
int main(void)
 time t t1, t3;
 struct tm *t2;
 t1 = time(NULL);
 t2 = localtime(&t1);
 t2 \rightarrow tm_mday += 40;
 t2 -> tm hour += 16;
 t3 = mktime(t2);
 printf("40 days and 16 hours from now, it will be a s \in n",
         wday[t2 -> tm_wday]);
/******* Output should be similar to: ********
40 days and 16 hours from now, it will be a Sunday
```

Related Information

- "asctime() Convert Time to Character String" on page 40
- "asctime_r() Convert Time to Character String (Restartable)" on page 42
- "ctime() Convert Time to Character String" on page 66
- "ctime_r() Convert Time to Character String (Restartable)" on page 67
- "gmtime() Convert Time" on page 141
- "gmtime_r() Convert Time (Restartable)" on page 143
- "localtime() Convert Time" on page 163
- "localtime_r() Convert Time (Restartable)" on page 164
- "time() Determine Current Time" on page 373
- "<time.h>" on page 17

modf() — Separate Floating-Point Value

Format

```
#include <math.h>
double modf(double x, double *intptr);
```

Language Level: ANSI

Description

The modf() function breaks down the floating-point value x into fractional and integral parts. The signed fractional portion of x is returned. The integer portion is stored as a double value pointed to by intptr. Both the fractional and integral parts are given the same sign as x.

Return Value

The modf() function returns the signed fractional portion of x.

Example that uses modf()

This example breaks the floating-point number -14.876 into its fractional and integral components.

```
#include <math.h>
#include <stdio.h>
int main(void)
  double x, y, d;
  x = -14.876;
  y = modf(x, &d);
  printf("x = %1f\n", x);
  printf("Integral part = %lf\n", d);
  printf("Fractional part = %lf\n", y);
/****** Output should be similar to: *********
x = -14.876000
Integral part = -14.000000
Fractional part = -0.876000
```

- "fmod() Calculate Floating-Point Remainder" on page 91
- "frexp() Separate Floating-Point Value" on page 112
- "ldexp() Multiply by a Power of Two" on page 156
- "<math.h>" on page 8

nl_langinfo() —Retrieve Locale Information

Format

```
#include <langinfo.h>
#include <nl types.h>
char *nl_langinfo(nl_item item);
```

Thread Safe: NO.

Description

The nl langinfo() function retrieves from the current locale the string that describes the requested information specified by item.

The retrieval of the following information from the current locale is supported:

CODESET	Explanation	
D_T_FMT	string for formatting date and time	
D_FMT	date format string	
T_FMT	time format string	
T_FMT_AMPM	a.m. or p.m. time format string	
AM_STR	Ante Meridian affix	
PM_STR	Post Meridian affix	
DAY_1	name of the first day of the week (for example, Sunday)	
DAY_2	name of the second day of the week (for example, Monday)	
DAY_3	name of the third day of the week (for example, Tuesday)	
DAY_4	name of the fourth day of the week (for example, Wednesday)	
DAY_5	name of the fifth day of the week (for example, Thursday)	
DAY_6	name of the sixth day of the week (for example, Friday)	
DAY_7	name of the seventh day of the week (for example, Saturday)	
ABDAY_1	abbreviated name of the first day of the week	
ABDAY_2	abbreviated name of the second day of the week	
ABDAY_3	abbreviated name of the third day of the week	
ABDAY_4	abbreviated name of the fourth day of the week	
ABDAY_5	abbreviated name of the fifth day of the week	
ABDAY_6	abbreviated name of the sixth day of the week	
ABDAY_7	abbreviated name of the seventh day of the week	
MON_1	name of the first month of the year	
MON_2	name of the second month of the year	
MON_3	name of the third month of the year	
MON_4	name of the fourth month of the year	

MON_5	name of the fifth month of the year	
MON_6	name of the sixth month of the year	
MON_7	name of the seventh month of the year	
MON_8	name of the eighth month of the year	
MON_9	name of the ninth month of the year	
MON_10	name of the tenth month of the year	
MON_11	name of the eleventh month of the year	
MON_12	name of the twelfth month of the year	
ABMON_1	abbreviated name of the first month of the year	
ABMON_2	abbreviated name of the second month of the year	
ABMON_3	abbreviated name of the third month of the year	
ABMON_4	abbreviated name of the fourth month of the year	
ABMON_5	abbreviated name of the fifth month of the year	
ABMON_6	abbreviated name of the sixth month of the year	
ABMON_7	abbreviated name of the seventh month of the year	
ABMON_8	abbreviated name of the eighth month of the year	
ABMON_9	abbreviated name of the ninth month of the year	
ABMON_10	abbreviated name of the tenth month of the year	
ABMON_11	abbreviated name of the eleventh month of the year	
ABMON_12	abbreviated name of the twelfth month of the year	
ERA	era description segments	
ERA_D_FMT	era date format string	
ERA_D_T_FMT	era date and time format string	
ERA_T_FMT	era time format string	
ALT_DIGITS	alternative symbols for digits	
RADIXCHAR	radix character	
THOUSEP	separator for thousands	
YESEXPR	affirmative response expression	
NOEXPR	negative response expression	
YESSTR	affirmative response for yes/no queries	
NOSTR	negative response for yes/no queries	
CRNCYSTR	currency symbol, preceded by '-' if the symbol should appear before the value, '+' if the symbol should appear after the value, or '.' if the symbol should replace the radix character	

Returned Value

The nl_langinfo() function returns a pointer to a null-ended string containing information concerning the active language or cultural area. The active language or cultural area is determined by the most recent setlocale() call. The array pointed to by the returned value is modified by subsequent calls to the function. The array should not be changed by the user's program.

If the item is not valid, the function returns a pointer to an empty string.

Example that uses nl langinfo()

This example retrieves the name of the codeset using the nl_langinfo() function.

```
#include <langinfo.h>
#include <locale.h>
#include <nl types.h>
#include <stdio.h>
int main(void)
 printf("Current codeset is %s\n", nl_langinfo(CODESET));
 return 0;
The output should be similar to:
 Current codeset is 37
```

Related Information

- "localeconv() Retrieve Information from the Environment" on page 158
- "setlocale() Set Locale" on page 308
- "<langinfo.h>" on page 7
- "<nl_types.h>" on page 9

perror() — Print Error Message

Format

```
#include <stdio.h>
void perror(const char *string);
```

Language Level: ANSI

Description

The perror() function prints an error message to stderr. If string is not NULL and does not point to a null character, the string pointed to by string is printed to the standard error stream, followed by a colon and a space. The message associated with the value in errno is then printed followed by a new-line character.

To produce accurate results, you should ensure that the perror() function is called immediately after a library function returns with an error; otherwise, subsequent calls may alter the errno value.

Return Value

There is no return value.

The value of errno may be set to:

Value Meaning

EBADDATA

The message data is not valid.

EBUSY

The record or file is in use.

ENOENT

The file or library cannot be found.

EPERM

Insufficient authorization for access.

ENOREC

Record not found.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

Example that uses perror()

This example tries to open a stream. If fopen() fails, the example prints a message and ends the program.

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *fh;
    if ((fh = fopen("mylib/myfile","r")) == NULL)
    {
        perror("Could not open data file");
        abort();
    }
}
```

Related Information

- "clearerr() Reset Error Indicators" on page 62
- "ferror() Test for Read/Write Errors" on page 79
- "strerror() Set Pointer to Run-Time Error Message" on page 333
- "<stdio.h>" on page 15

pow() — Compute Power

Format

```
#include <math.h>
double pow(double x, double y);
```

Language Level: ANSI

Description

The pow() function calculates the value of x to the power of y.

Return Value

If y is 0, the pow() function returns the value 1. If x is 0 and y is negative, the pow() function sets errno to EDOM and returns 0. If both x and y are 0, or if x is negative and y is not an integer, the pow() function sets errno to EDOM, and

returns 0. The errno variable may also be set to ERANGE. If an overflow results, the pow() function returns +HUGE_VAL for a large result or -HUGE_VAL for a small result.

Example that uses pow()

This example calculates the value of 2^3 .

```
#include <math.h>
#include <stdio.h>
int main(void)
  double x, y, z;
  x = 2.0;
  y = 3.0;
  z = pow(x,y);
  printf("% If to the power of % If is % If \n", x, y, z);
/****** Output should be similar to: *********
2.000000 to the power of 3.000000 is 8.000000
```

Related Information

- "exp() Calculate Exponential Function" on page 73
- "log() Calculate Natural Logarithm" on page 165
- "log10() Calculate Base 10 Logarithm" on page 166
- "sqrt() Calculate Square Root" on page 318
- "<math.h>" on page 8

printf() — Print Formatted Characters

Format

```
#include <stdio.h>
int printf(const char *format-string, argument-list);
```

Language Level: ANSI

Thread Safe: YES

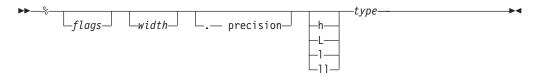
Description

The printf() function formats and prints a series of characters and values to the standard output stream stdout. Format specifications, beginning with a percent sign (%), determine the output format for any argument-list following the format-string. The format-string is a multibyte character string beginning and ending in its initial shift state.

The format-string is read left to right. When the first format specification is found, the value of the first argument after the format-string is converted and output according to the format specification. The second format specification causes the second argument after the format-string to be converted and output, and so on through the end of the format-string. If there are more arguments than there are format specifications, the extra arguments are evaluated and ignored. The results

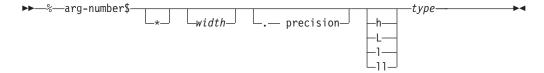
are undefined if there are not enough arguments for all the format specifications. Only 15 significant digits are guaranteed for conversions of floating point numbers.

A format specification has the following form:



Conversions can be applied to the nth argument after the *format-string* in the argument list, rather than to the next unused argument. In this case, the conversion character % is replaced by the sequence %n\$, where n is a decimal integer in the range 1 thru NL_ARGMAX, giving the position of the argument in the argument list. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages.

Alternative format specification has the following form:



As an alternative, specific entries in the argument-list may be assigned by using the format specification outlined in the diagram above. This format specification and the previous format specification may not be mixed in the same call to scanf(). Otherwise, unpredictable results may occur.

The arg-number is a positive integer constant where 1 refers to the first entry in the argument-list. Arg-number may not be greater than the number of entries in the argument-list, or else the results are undefined. Arg-number also may not be greater than NL_ARGMAX.

In format strings containing the %n\$ form of conversion specifications, numbered arguments in the argument list can be referenced from the format string as many times as required.

In format strings containing the %n\$ form of a conversion specification, a field width or precision may be indicated by the sequence *m\$, where m is a decimal integer in the range 1 thru NL_ARGMAX giving the position in the argument list (after the format argument) of an integer argument containing the field width or precision, for example:

```
printf("%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```

The format-string can contain either numbered argument specifications (that is, %n\$ and *m\$), or unnumbered argument specifications (that is, % and *), but normally not both. The only exception to this is that %% can be mixed with the %n\$ form. The results of mixing numbered and unnumbered argument specifications in a format-string string are undefined. When numbered argument specifications are used, specifying the nth argument requires that all the leading arguments, from the first to the (n-1)th, are specified in the format string.

Each field of the format specification is a single character or number signifying a particular format option. The type character, which appears after the last optional format field, determines whether the associated argument is interpreted as a character, a string, a number, or pointer. The simplest format specification contains only the percent sign and a type character (for example, %s).

The following optional fields control other aspects of the formatting:

Field Description

Justification of output and printing of signs, blanks, decimal points, octal, flags and hexadecimal prefixes, and the semantics for wchar_t precision unit.

width Minimum number of bytes output.

precision

See Table 4 on page 206.

h, l, ll, L

Size of argument expected:

- A prefix with the integer types d, i, o, u, x, X, and n that specifies that the argument is short int or unsigned short int.
- 1 A prefix with d, i, o, u, x, X, and n types that specifies that the argument is a long int or unsigned long int.
- 11 A prefix with d, i, o, u, x, X, and n types that specifies that the argument is a long long int or unsigned long long int.
- L A prefix with e, E, f, g, or G types that specifies that the argument is long double.

Each field of the format specification is discussed in detail below. If a percent sign (%) is followed by a character that has no meaning as a format field, the character is simply copied to stdout. For example, to print a percent sign character, use %%.

The *type* characters and their meanings are given in the following table:

Character	Argument	Output Format
d, i	Integer	Signed decimal integer.
u	Integer	Unsigned decimal integer.
o	Integer	Unsigned octal integer.
X	Integer	Unsigned hexadecimal integer, using abcdef.
X	Integer	Unsigned hexadecimal integer, using ABCDEF.
D(n,p)	Packed decimal	It has the format [-] <code>dddd.dddd</code> where the number of digits after the decimal point is equal to the precision of the specification. If the precision is missing, the default is p; if the precision is zero, and the # flag is not specified, no decimal point character appears. If the n and the p are *, an argument from the argument list supplies the value. n and p must precede the value being formatted in the argument list. At least one character appears before a decimal point. The value is rounded to the appropriate number of digits.

Character	Argument	Output Format
f	Double	Signed value having the form [-]dddd.dddd, where dddd is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number. The number of digits after the decimal point is equal to the requested precision.
e	Double	Signed value having the form [-] <i>d.dddd</i> e[<i>sign</i>] <i>ddd</i> , where <i>d</i> is a single-decimal digit, <i>dddd</i> is one or more decimal digits, <i>ddd</i> is 2 or 3 decimal digits, and <i>sign</i> is + or
Е	Double	Identical to the e format except that E introduces the exponent instead of e.
g	Double	Signed value printed in f or e format. The e format is used only when the exponent of the value is less than -4 or greater than <i>precision</i> . Trailing zeros are truncated, and the decimal point appears only if one or more digits follow it.
G	Double	Identical to the g format except that E introduces the exponent (where appropriate) instead of e.
С	Character (byte)	Single character.
S	String	Characters (bytes) printed up to the first null character (\0) or until <i>precision</i> is reached. ¹
n	Pointer to integer	Number of characters (bytes) successfully written so far to the <i>stream</i> or buffer; this value is stored in the integer whose address is given as the argument. ¹
p	Pointer	Pointer converted to a sequence of printable characters. It can be one of the following: • space pointer • system pointer • invocation pointer • procedure pointer
		 open pointer suspend pointer data pointer label pointer
lc or C	Wide Character	The (wchar_t) character is converted to a multibyte character as if by a call to wctomb(), and this character is printed out. ²
ls or S	Wide Character	The (wchar_t) characters up to the first (wchar_t) null character (L\0), or until precision is reached, are converted to multibyte characters, as if by a call to wcstombs(), and these characters are printed out. If the argument is a null string, (null) is printed. ²

Note¹ If the program is compiled with LOCALETYPE(*LOCALEUCS2) and SYSIFCOPT(*IFSIO), then the wide characters are assumed to be UNICODE characters.

Note² When printing to the screen in a UNICODE environment, the wide characters are converted to the CCSID of the job before they are printed out.

The following list shows the format of the printed values for iSeries pointers, and gives a brief description of the components of the printed values.

Space pointer: SPP:Context:Object:Offset:AG

Context: type, subtype and name of the context Object: type, subtype and name of the object

Offset: offset within the space AG: Activation group ID

System pointer: SYP:Context:Object:Auth:Index:AG

Context: type, subtype and name of the context Object: type, subtype and name of the object

Auth: authority

Index: Index associated with the pointer

AG: Activation group ID

Invocation pointer: IVP:Index:AG

Index associated with the pointer

AG: Activation group ID

Procedure pointer: PRP:Index:AG

Index: Index associated with the pointer

AG: Activation group ID

Suspend pointer: SUP:Index:AG

Index: Index associated with the pointer

AG: Activation group ID

Data pointer: DTP:Index:AG

Index: Index associated with the pointer

AG: Activation group ID

Label pointer: LBP:Index:AG

Index: Index associated with the pointer

AG: Activation group ID

NULL pointer: NULL

The following restrictions apply to pointer printing and scanning on the iSeries

- If a pointer is printed out and scanned back from the same activation group, the scanned back pointer will be compared equal to the pointer printed out.
- If a scanf() family function scans a pointer that was printed out by a different activation group, the scanf() family function will set the pointer to NULL.

See the WebSphere Development Studio: ILE C/C++ Programmer's Guide for more information about using iSeries pointers.

The *flag* characters and their meanings are as follows (notice that more than one *flag* can appear in a format specification):

Flag	Meaning	Default
-	Left-justify the result within the field width.	Right-justify.
+	Prefix the output value with a sign (+ or –) if the output value is of a signed type.	Sign appears only for negative signed values (–).
blank(' ')	Prefix the output value with a blank if the output value is signed and positive. The + flag overrides the <i>blank</i> flag if both appear, and a positive signed value will be output with a sign.	No blank.
#	When used with the o, x, or X formats, the # flag prefixes any nonzero output value with 0, 0x, or 0X, respectively.	No prefix.
	When used with the f, D(n,p), e, or E formats, the # flag forces the output value to contain a decimal point in all cases.	Decimal point appears only if digits follow it.
	When used with the g or G formats, the # flag forces the output value to contain a decimal point in all cases and prevents the truncation of trailing zeros.	Decimal point appears only if digits follow it; trailing zeros are truncated.
	When used with the ls format, the # flag causes precision to be measured in characters, regardless of the size of the character. For example, if single-byte characters are being printed, a precision of 4 would result in 4 bytes being printed. If double-byte characters are being printed, a precision of 4 would result in 8 bytes being printed.	Precision indicates the maximum number of bytes to be output.
	When used with the p format, the # flag converts the pointer to hex digits. These hex digits cannot be converted back into a pointer.	Pointer converted to a sequence of printable characters.
0	When used with the d, i, D(n,p) o, u, x, X, e, E, f, g, or G formats, the 0 flag causes leading 0s to pad the output to the field width. The 0 flag is ignored if precision is specified for an integer or if the – flag is specified.	Space padding. No space padding for D(n,p).

The # flag should not be used with c, lc, d, i, u, or s types.

Width is a nonnegative decimal integer controlling the minimum number of characters printed. If the number of characters (bytes) in the output value is less than the specified *width*, blanks are added on the left or the right (depending on whether the - flag is specified) until the minimum width is reached.

Width never causes a value to be truncated; if the number of characters (bytes) in the output value is greater than the specified *width*, or *width* is not given, all characters of the value are printed (subject to the *precision* specification).

For the ls type, width is specified in bytes. If the number of bytes in the output value is less than the specified width, single-byte blanks are added on the left or the right (depending on whether the - flag is specified) until the minimum width is reached.

The width specification can be an asterisk (*), in which case an argument from the argument list supplies the value. The width argument must precede the value being formatted in the argument list.

Precision is a nonnegative decimal integer preceded by a period, which specifies the number of characters to be printed or the number of decimal places. Unlike the width specification, the precision can cause truncation of the output value or rounding of a floating-point or packed decimal value.

The precision specification can be an asterisk (*), in which case an argument from the argument list supplies the value. The precision argument must precede the value being formatted in the argument list.

The interpretation of the *precision* value and the default when the *precision* is omitted depend on the *type*, as shown in the following table:

Table 4. Values of Precision

Type	Meaning	Default
i d u o x X	Precision specifies the minimum number of digits to be printed. If the number of digits in the argument is less than precision, the output value is padded on the left with zeros. The value is not truncated when the number of digits exceeds precision.	If precision is 0 or omitted entirely, or if the period (.) appears without a number following it, the precision is set to 1.
f D(n,p) e E	Precision specifies the number of digits to be printed after the decimal point. The last digit printed is rounded.	Default <i>precision</i> for f, e and E is six. Default <i>precision</i> for D(n,p) is p. If <i>precision</i> is 0 or the period appears without a number following it, no decimal point is printed.
g G	<i>Precision</i> specifies the maximum number of significant digits printed.	All significant digits are printed. Default <i>precision</i> is six.
С	No effect.	The character is printed.
lc	No effect.	The wchar_t character is converted and resulting mulitbyte character is printed.
S	<i>Precision</i> specifies the maximum number of characters to be printed. Characters in excess of <i>precision</i> are not printed.	Characters are printed until a null character is encountered.
ls	Precision specifies the maximum number of bytes to be printed. Bytes in excess of precision are not printed; however, multibyte integrity is always preserved.	wchar_t characters are converted and resulting multibyte characters are printed.

Return Value

The printf() function returns the number of bytes printed. The value of errno may be set to ETRUNC.

Note: The radix character for the printf() function is locale sensitive. The radix character is the decimal point to be used for the #flag character of the format string parameter for the format types f, D(n,p), e, E, g, and G.

Example that uses printf()

This example prints data in a variety of formats.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
   char ch = 'h', *string = "computer";
   int count = 234, hex = 0x10, oct = 010, dec = 10;
   double fp = 251.7366;
   wchar_t wc = (wchar_t)0x0058;
   wchar t ws[4];
   printf("1234567890123%n4567890123456789\n\n", &count);
   printf("Value of count should be 13; count = %d\n\n", count);
   printf("%10c%5c\n", ch, ch);
   printf("%25s\n%25.4s\n\n", string, string);
   printf("%f
                             %E\n\n", fp, fp, fp, fp);
               %.2f %e
   printf("%i
                      %i\n\n", hex, oct, dec);
                %i
/****** Output should be similar to: *********
234
     +234
             000234
                       FΑ
                             ea
                                   352
12345678901234567890123456789
Value of count should be 13; count = 13
 h
      h
         computer
             comp
251.736600
             251.74
                      2.517366e+02
                                     2.517366E+02
     8
           10
16
Example that uses printf()
    #include <stdio.h>
    #include <stdlib.h>
    #include <locale.h>
                                                                         */
    /* This program is compiled with LOCALETYPE(*LOCALEUCS2) and
    /* SYSIFCOPT(*IFSIO)
    /* We will assume the locale setting is the same as the CCSID of the */
    /* job. We will also assume any files involved have a CCSID of
    /* 65535 (no convert). This way if printf goes to the screen or
    /* a file the output will be the same.
    int main(void)
       wchar t wc = 0x0058;
                             /* UNICODE X */
       wchar t ws[4];
       setlocale(LC ALL,
        "/QSYS.LIB/EN US.LOCALE"); /* a CCSID 37 locale */
```

```
ws[0] = 0x0041;
                              /* UNICODE A */
       ws[1] = (wchar_t)0x0042; /* UNICODE B
       ws[2] = (wchar_t)0x0043;
ws[3] = (wchar_t)0x0000;
                                       /* UNICODE C
       /* The output displayed is CCSID 37 */
       printf("%lc %ls\n\n",wc,ws);
       printf("%lc %.2ls\n\n",wc,ws);
       /* Now lets try a mixed byte CCSID example */
       /* You would need a device that can handle mixed bytes to */
       /* display this correctly.
        setlocale(LC ALL,
        "/QSYS.LIB/JA_JP.LOCALE");/* a CCSID 5026 locale */
       /* big A means an A that takes up 2 bytes on the screen
        /* It will look bigger then single byte A
       ws[0] = (wchar_t)0xFF21; /* UNICODE big A
                                                         */
       ws[1] = (wchar_t)0xFF22;
                                      /* UNICODE big B
       ws[2] = (wchar_t)0xFF23;
                                      /* UNICODE big C
       ws[3] = (wchar_t)0x0000;
       wc = 0xff11;
                                       /* UNICODE big 1 */
       printf("%lc %ls\n\n",wc,ws);
       /* The output of this printf is not shown below and it */
       /* will differ depending on the device you display it */
       /* but if you looked at the string in hex it would look */
       /* like this: 0E42F10F404040400E42C142C242C30F
       /* OE is shift out, OF is shift in, and 42F1 is the
       /* big 1 in CCSID 5026 */
       printf("%lc %.4ls\n\n",wc,ws);
       /* The output of this printf is not shown below either. */
       /* The hex would look like:
                                                               */
       /* 0E42F10F404040400E42C10F
       /* Since the precision is in bytes we only get 4 bytes */
       /* of the string.
       printf("%lc %#.21s\n\n",wc,ws);
       /* The output of this printf is not shown below either. */
       /* The hex would look like:
                                                               */
       /* 0E42F10F404040400E42C142C20F
                                                               */
       /* The # means precision is in characters reguardless
       /* of size. So we get 2 characters of the string.
/******* Output should be similar to: ********
χ
      ABC
Χ
      ΔR
Example that uses printf()
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
/* This program is compile LOCALETYPE(*LOCALE) and
/* SYSIFCOPT(*IFSIO)
                      */
int main(void)
```

```
wchar t wc = (wchar t)0x00C4;
    wchar_t ws[4];
ws[0] = (wchar_t)0x00C1;
    ws[1] = (wchar_t)0x00C2;
                                   /* B */
    ws[2] = (wchar_t)0x00C3;
    ws[3] = (wchar t)0x0000;
    /* The output displayed is CCSID 37 */
    printf("%lc %ls\n\n",wc,ws);
    /* Now lets try a mixed byte CCSID example */
    /* You would need a device that can handle mixed bytes to */
    /* display this correctly.
    setlocale(LC ALL,
    "/QSYS.lib/JA JP.LOCALE"); /* a CCSID 5026 locale */
    /* big A means an A that takes up 2 bytes on the screen
    /* It will look bigger then single byte A
    ws[0] = (wchar_t)0x42C1;
                                   /* big A
                                              */
    ws[1] = (wchar_t)0x42C2;
ws[2] = (wchar_t)0x42C3;
                                    /* big B
                                   /* big C
    ws[3] = (wchar_t)0x0000;
    wc = 0x42F1;
                                    /* big 1 */
    printf("%lc %ls\n\n",wc,ws);
    /* The output of this printf is not shown below and it */
    /* will differ depending on the device you display it */
    /* but if you looked at the string in hex it would look */
    /* like this: 0E42F10F404040400E42C142C242C30F
    /* OE is shift out, OF is shift in, and 42F1 is the
                                                           */
    /* big 1 in CCSID 5026 */
    printf("%lc %.4ls\n\n",wc,ws);
    /* The output of this printf is not shown below either. */
    /* The hex would look like:
    /* 0E42F10F404040400E42C10F
    /* Since the precision is in bytes we only get 4 bytes */
    /* of the string.
    printf("%lc %#.21s\n\n",wc,ws);
    /* The output of this printf is not shown below either. */
    /* The hex would look like:
    /* 0E42F10F404040400E42C142C20F
    /* The # means precision is in characters regardless
    /* of size. So we get 2 characters of the string.
/****** Output should be similar to: *********
n
      ARC
```

- "fprintf() Write Formatted Data to a Stream" on page 99
- "fscanf() Read Formatted Data" on page 113
- "scanf() Read Data" on page 299
- "sprintf() Print Formatted Data to Buffer" on page 317

- "sscanf() Read Data" on page 322
- "vfprintf() Print Argument Data to Stream" on page 386
- "vprintf() Print Argument Data" on page 389
- "vsprintf() Print Argument Data to Buffer" on page 391
- "wprintf() Format Data as Wide Characters and Print" on page 447
- "<stdio.h>" on page 15

putc() - putchar() — Write a Character

Format

```
#include <stdio.h>
int putc(int c, FILE *stream);
int putchar(int c);
```

Language Level: ANSI

Thread Safe: NO. #undef putc or #undef putchar allows the putc or putchar function to be called instead of the macro version of these functions. The functions are thread safe.

Description

The putc() function converts *c* to unsigned char and then writes *c* to the output stream at the current position. The putchar() is equivalent to putc(c, stdout).

The putc() function can be defined as a macro so the argument can be evaluated multiple times.

The putc() and putchar() functions are not supported for files opened with type=record.

Return Value

The putc() and putchar() functions return the character written. A return value of EOF indicates an error.

The value of errno may be set to:

Value Meaning

EPUTANDGET

An illegal write operation occurred after a read operation.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

Example that uses putc()

This example writes the contents of a buffer to a data stream. In this example, the body of the for statement is null because the example carries out the writing operation in the test expression.

- "fputc() Write Character" on page 101
- "fwrite() Write Items" on page 127
- "getc() getchar() Read a Character" on page 133
- "puts() Write a String" on page 212
- "putwc() Write Wide Character" on page 213
- "putwchar() Write Wide Character to stdout" on page 214
- "<stdio.h>" on page 15

putenv() — Change/Add Environment Variables

Format

```
#include <stdlib.h>
int *putenv(const char *varname);
```

Description

The putenv() function sets the value of an environment variables by altering an existing variable or creating a new one. The *varname* parameter points to a string of the form var=x, where x is the new value for the environment variable var.

The name cannot contain a blank or an equal (=) symbol. For example, PATH NAME=/my_lib/joe_user

is not valid because of the blank between PATH and NAME. Similarly, PATH=NAME=/my_lib/joe_user

is not valid because of the equal symbol between PATH and NAME. The system interprets all characters following the first equal symbol as being the value of the environment variable.

Return Value

The putenv() function returns 0 is successful. If putenv() fails then -1 is returned and errno is set to indicate the error.

Example that uses putenv()

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
  char *pathvar;
  if (-1 == putenv("PATH=/:/home/userid")) {
     printf("putenv failed \n");
     return EXIT FAILURE;
  /* getting and printing the current environment path
                                                      */
  pathvar = getenv("PATH");
  printf("The current path is: %s\n", pathvar);
  return 0;
/*****************
  The output should be:
```

The current path is: /:/home/userid

Related Information

- "getenv() Search for Environment Variables" on page 135
- "<stdlib.h>" on page 16

puts() — Write a String

Format

```
#include <stdio.h>
int puts(const char *string);
```

Language Level: ANSI

Thread Safe: YES

Description

The puts() function writes the given string to the standard output stream stdout; it also appends a new-line character to the output. The ending null character is not written.

Return Value

The puts() function returns EOF if an error occurs. A nonnegative return value indicates that no error has occurred.

Example that uses puts()

This example writes Hello World to stdout.

```
#include <stdio.h>
int main(void)
{
  if ( puts("Hello World") == EOF )
    printf( "Error in puts\n" );
}
/**********************
Hello World
*/
```

- "fputs() Write String" on page 103
- "fputws() Write Wide-Character String" on page 106
- "gets() Read a Line" on page 137
- "putc() putchar() Write a Character" on page 210
- "putwc() Write Wide Character"
- "<stdio.h>" on page 15

putwc() — Write Wide Character

I

Format

```
#include <stdio.h>
#include <wchar.h>
wint_t putwc(wint_t wc, FILE *stream);
```

Language Level: ANSI

Thread Safe: YES

Description

The putwc() function converts the wide character wc to a multibyte character, and writes it to the stream at the current position. It also advances the file position indicator for the stream appropriately. The putwc() function is equivalent to the fputwc() function except that some platforms implement putwc() as a macro. Therefore, for portability, the stream argument to putwc() should not be an expression with side effects.

The behavior of the putwc() function is affected by the LC_CTYPE category of the current locale. Using a non-wide-character function with the putwc() function on the same stream results in undefined behavior. After calling the putwc() function, flush the buffer or reposition the stream pointer before calling a write function for the stream, unless EOF has been reached. After a write operation on the stream, flush the buffer or reposition the stream pointer before calling the putwc() function.

Note: This function is available when SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) are specified on the compilation command.

Return Value

The putwc() function returns the wide character written. If a write error occurs, it sets the error indicator for the stream and returns WEOF. If an encoding error

occurs when a wide character is converted to a multibyte character, the putwc() function sets errno to EILSEQ and returns WEOF.

Example that uses putwc()

The following example uses the putwc() function to convert the wide characters in wcs to multibyte characters and write them to the file **putwc.out**.

```
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <errno.h>
int main(void)
  FILE
         *stream;
  wchar_t *wcs = L"A character string.";
  int.
         i;
  if (NULL == (stream = fopen("putwc.out", "w"))) {
     printf("Unable to open: \"putwc.out\".\n");
     exit(1);
  }
  for (i = 0; wcs[i] != L' \setminus 0'; i++) {
     errno = 0;
     if (WEOF == putwc(wcs[i], stream)) {
       printf("Unable to putwc() the wide character.\n"
              "wcs[%d] = 0x%1x\n", i, wcs[i]);
        if (EILSEQ == errno)
          printf("An invalid wide character was encountered.\n");
       exit(1);
  fclose(stream);
  return 0;
  /***********************************
     The output file putwc.out should contain:
     A character string.
```

Related Information

- "fputc() Write Character" on page 101
- "fputwc() Write Wide Character" on page 104
- "fputws() Write Wide-Character String" on page 106
- "getwc() Read Wide Character from Stream" on page 138
- "putc() putchar() Write a Character" on page 210
- "putwchar() Write Wide Character to stdout"
- "<stdio.h>" on page 15
- "<wchar.h>" on page 18

putwchar() — Write Wide Character to stdout

Format

```
#include <wchar.h>
wint t putwchar(wint t wc);
```

Language Level: ANSI

Thread Safe: YES

Description

The putwchar() function converts the wide character wc to a multibyte character and writes it to stdout. A call to the putwchar() function is equivalent to putwc(wc, stdout).

The behavior of this function is affected by the LC_CTYPE category of the current locale. Using a non-wide-character function with the putwchar() function on the same stream results in undefined behavior. After calling the putwchar() function, flush the buffer or reposition the stream pointer before calling a write function for the stream, unless EOF has been reached. After a write operation on the stream, flush the buffer or reposition the stream pointer before calling the putwchar() function.

Note: This function is available when SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) are specified on the compilation command.

Return Value

The putwchar() function returns the wide character written. If a write error occurs, the putwchar() function sets the error indicator for the stream and returns WEOF. If an encoding error occurs when a wide character is converted to a multibyte character, the putwchar() function sets error to EILSEQ and returns WEOF.

Example that uses putwchar()

This example uses the putwchar() function to write the string in wcs.

```
#include <stdio.h>
#include <wchar.h>
#include <errno.h>
#include <stdlib.h>
int main(void)
  wchar t *wcs = L"A character string.";
         i;
  int
  for (i = 0; wcs[i] != L' \setminus 0'; i++) {
     errno = 0;
     if (WEOF == putwchar(wcs[i])) {
        printf("Unable to putwchar() the wide character.\n");
        printf("wcs[%d] = 0x%lx\n", i, wcs[i]);
        if (EILSEQ == errno)
          printf("An invalid wide character was encountered.\n");
        exit(EXIT FAILURE);
  return 0;
     The output should be similar to :
     A character string.
```

- "fputc() Write Character" on page 101
- "fputwc() Write Wide Character" on page 104
- "fputws() Write Wide-Character String" on page 106
- "getwchar() Get Wide Character from stdin" on page 140
- "putc() putchar() Write a Character" on page 210
- "putwc() Write Wide Character" on page 213
- "<wchar.h>" on page 18

qsort() — Sort Array

Format

```
#include <stdlib.h>
void qsort(void *base, size_t num, size_t width,
           int(*compare)(const void *key, const void *element));
```

Language Level: ANSI

Description

The qsort () function sorts an array of *num* elements, each of *width* bytes in size. The base pointer is a pointer to the array to be sorted. The qsort() function overwrites this array with the sorted elements.

The compare argument is a pointer to a function you must supply that takes a pointer to the key argument and to an array element, in that order. The qsort() function calls this function one or more times during the search. The function must compare the key and the element and return one of the following values:

Value	Meaning
	11204111119

Less than 0	key less than element
0	key equal to element
Greater than 0	key greater than element

Value Meaning

Less than 0

key less than element

key equal to element

Greater than 0

key greater than element

The sorted array elements are stored in ascending order, as defined by your compare function. You can sort in reverse order by reversing the sense of "greater than" and "less than" in compare. The order of the elements is unspecified when two elements compare equally.

Return Value

There is no return value.

Example that uses qsort()

This example sorts the arguments (argv) in ascending lexical sequence, using the comparison function compare() supplied in the example.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
              /* Declaration of compare() as a function */
int compare(const void *, const void *);
int main (int argc, char *argv[])
  int i;
  argv++;
  argc--;
  qsort((char *)argv, argc, sizeof(char *), compare);
  for (i = 0; i < argc; ++i)
     printf("%s\n", argv[i]);
  return 0;
int compare (const void *arg1, const void *arg2)
              /* Compare all of both strings */
  return(strcmp(*(char **)arg1, *(char **)arg2));
/****** If the program is passed the arguments: *******
****** 'Does' 'this' 'really' 'sort' 'the' arguments' 'correctly?'****
****** then the expected output is: *********
arguments
correctly?
really
sort
the
this
Does
*/
```

- "bsearch() Search Arrays" on page 52
- "<stdlib.h>" on page 16

QXXCHGDA() —Change Data Area

Format

```
#include <xxdtaa.h>
void QXXCHGDA( DTAA NAME T dtaname, short int offset, short int len,
              char *dtaptr);
```

Language Level: ILE C Extension

Description

The QXXCHGDA() function allows you to change the data area specified by dtaname, starting at position offset, with the data in the user buffer pointed to by dtaptr of length len. The structure dtaname contains the names of the data area and the library that contains the data area. The values that can be specified for the data area name are:

*LDA Specifies that the contents of the local data area are to be changed. The library name dtaa_lib must be blank.

*GDA Specifies that the contents of the group data area are to be changed. The library name *dtaa_lib* must be blank.

data-area-name

Specifies that the contents of the data area created using the Create Data Area (CRTDTAARA) CL command are to be changed. The library name *dtaa_lib* must be either *LIBL, *CURLIB, or the name of the library where the data area (*data-area-name*) is located. The data area is locked while it is being changed.

QXXCHGDA can only be used to change character data.

Example that uses QXXCHGDA()

Related Information

• "QXXRTVDA() —Retrieve Data Area" on page 223

QXXDTOP() — Convert Double to Packed Decimal

Format

Language Level: ILE C Extension

Description

The QXXDTOP function converts the *double* value specified in *value* to a packed decimal number with *digits* total digits, and *fraction* fractional digits. The result is stored in the array pointed to by *zptr*.

Example that uses QXXDTOP()

```
#include <xxcvt.h>
#include <stdio.h>
int main(void)
 unsigned char pptr[10];
 int digits = 8, fraction = 6;
 double value = 3.141593;
 QXXDTOP(pptr, digits, fraction, value);
```

- "QXXDTOZ() —Convert Double to Zoned Decimal"
- "QXXITOP() —Convert Integer to Packed Decimal" on page 221
- "QXXITOZ() —Convert Integer to Zoned Decimal" on page 221
- "QXXPTOD() —Convert Packed Decimal to Double" on page 222
- "QXXPTOI() —Convert Packed Decimal to Integer" on page 223
- "QXXZTOD() —Convert Zoned Decimal to Double" on page 224
- "QXXZTOI() —Convert Zoned Decimal to Integer" on page 225

QXXDTOZ() —Convert Double to Zoned Decimal

Format

```
#include <xxcvt.h>
void QXXDTOZ(unsigned char *zptr, int digits, int fraction,
             double value);
```

Language Level: ILE C Extension

Description

The QXXDTOZ function converts the double value specified in value to a zoned decimal number with digits total digits, and fraction fractional digits. The result is stored in the array pointed to by zptr.

Example that uses QXXDTOZ()

```
#include <xxcvt.h>
#include <stdio.h>
int main(void)
 unsigned char zptr[10];
 int digits = 8, fraction = 6;
 double value = 3.141593;
 QXXDTOZ(zptr, digits, fraction, value);
                          /* Zoned value is : 03141593 */
```

Related Information

- "QXXDTOP() Convert Double to Packed Decimal" on page 219
- "QXXITOP() —Convert Integer to Packed Decimal" on page 221
- "QXXITOZ() —Convert Integer to Zoned Decimal" on page 221
- "QXXPTOD() —Convert Packed Decimal to Double" on page 222
- "QXXPTOI() —Convert Packed Decimal to Integer" on page 223

- "QXXZTOD() —Convert Zoned Decimal to Double" on page 224
- "QXXZTOI() —Convert Zoned Decimal to Integer" on page 225

QXXITOP() —Convert Integer to Packed Decimal

Format

Language Level: ILE C Extension

Description

The QXXITOP function converts the integer specified in *value* to a packed decimal number with *digits* total digits, and *fraction* fractional digits. The result is stored in the array pointed to by *pptr*.

Example that uses QXXITOP()

```
#include <xxcvt.h>
#include <stdio.h>

int main(void)
{
   unsigned char pptr[10];
   int digits = 3, fraction = 0;
   int value = 116;

   QXXITOP(pptr, digits, fraction, value);
}
```

Related Information

- "QXXDTOP() Convert Double to Packed Decimal" on page 219
- "QXXDTOZ() —Convert Double to Zoned Decimal" on page 220
- "QXXITOZ() —Convert Integer to Zoned Decimal"
- "QXXPTOD() —Convert Packed Decimal to Double" on page 222
- "QXXPTOI() —Convert Packed Decimal to Integer" on page 223
- "QXXZTOD() —Convert Zoned Decimal to Double" on page 224
- "QXXZTOI() —Convert Zoned Decimal to Integer" on page 225

QXXITOZ() —Convert Integer to Zoned Decimal

Format

```
#include <xxcvt.h>
void QXXITOZ(unsigned char *zptr, int digits, int fraction, int value);
```

Language Level: ILE C Extension

Description

The QXXITOZ function converts the integer specified in *value* to a zoned decimal number with *digits* total digits, and *fraction* fractional digits. The result is stored in the array pointed to by *zptr*.

Example that uses QXXITOZ()

```
#include <xxcvt.h>
#include <stdio.h>
int main(void)
 unsigned char zptr[10];
 int digits = 9, fraction = 0;
 int value = 111115;
 QXXITOZ(zptr, digits, fraction, value);
                        /* Zoned value is : 000111115 */
```

Related Information

- "QXXDTOP() Convert Double to Packed Decimal" on page 219
- "QXXDTOZ() —Convert Double to Zoned Decimal" on page 220
- "QXXITOP() —Convert Integer to Packed Decimal" on page 221
- "QXXPTOD() —Convert Packed Decimal to Double"
- "QXXPTOI() —Convert Packed Decimal to Integer" on page 223
- "QXXZTOD() —Convert Zoned Decimal to Double" on page 224
- "QXXZTOI() —Convert Zoned Decimal to Integer" on page 225

QXXPTOD() —Convert Packed Decimal to Double

Format

```
#include <xxcvt.h>
double QXXPTOD(unsigned char *pptr, int digits, int fraction);
```

Language Level: ILE C Extension

Description

The QXXPTOD function converts a packed decimal number to a double.

Example that uses QXXPTOD()

```
#include <xxcvt.h>
#include <stdio.h>
int main(void)
 unsigned char pptr[10];
 int digits = 8, fraction = 6;
 double value = 6.123456, result;
          /* First convert an integer to a packed decimal,*/
 QXXDTOP(pptr, digits, fraction, value);
         /* then convert it back to a double.
                                                           */
 result = QXXPTOD(pptr, digits, fraction);
         /* result = 6.123456
```

Related Information

- "QXXDTOP() Convert Double to Packed Decimal" on page 219
- "QXXDTOZ() —Convert Double to Zoned Decimal" on page 220
- "QXXITOP() —Convert Integer to Packed Decimal" on page 221

- "QXXITOZ() —Convert Integer to Zoned Decimal" on page 221
- "QXXPTOI() —Convert Packed Decimal to Integer"
- "QXXZTOD() —Convert Zoned Decimal to Double" on page 224
- "QXXZTOI() —Convert Zoned Decimal to Integer" on page 225

QXXPTOI() —Convert Packed Decimal to Integer

Format

```
#include <xxcvt.h>
int QXXPTOI(unsigned char *pptr, int digits, int fraction);
```

Language Level: ILE C Extension

Description

The QXXPTOI function converts a packed decimal number to an integer.

Example that uses QXXPTOI()

Related Information

- "QXXDTOP() Convert Double to Packed Decimal" on page 219
- "QXXDTOZ() —Convert Double to Zoned Decimal" on page 220
- "QXXITOP() —Convert Integer to Packed Decimal" on page 221
- "QXXITOZ() —Convert Integer to Zoned Decimal" on page 221
- "QXXPTOD() —Convert Packed Decimal to Double" on page 222
- "QXXZTOD() —Convert Zoned Decimal to Double" on page 224
- "QXXZTOI() —Convert Zoned Decimal to Integer" on page 225

QXXRTVDA() —Retrieve Data Area

Format

Language Level: ILE C Extension

Description

The following typedef definition is included in the <xxdtaa.h> header file. The character arrays are not null-ended strings so they must be blank filled.

```
typedef struct DTAA NAME T {
   char dtaa name[10]; /* name of data area */
   char dtaa lib[10]; /* library that contains data area */
}_DTAA_NAME_T;
```

The QXXRTVDA function retrieves a copy of the data area specified by dtaname starting at position offset with a length of len. The structure dtaname contains the names of the data area and the library that contains the data area. The values that can be specified for the data area name are:

- *LDA The contents of the local data area are to be retrieved. The library name dtaa lib must be blank.
- *GDA The contents of the group data area are to be retrieved. The library name dtaa lib must be blank.
- *PDA Specifies that the contents of the program initialization parameters (PIP) data area are to be retrieved. The PIP data area is created for each pre-started job and is a character area up to 2000 characters in length. You cannot retrieve the PIP data area until you have acquired the requester. The library name *dtaa_lib* must be blank.

data-area-name

Specifies that the contents of the data area created using the Create Data Area (CRTDTAARA) CL command are to be retrieved. The library name dtaa_lib must be either *LIBL, *CURLIB, or the name of the library where the data area (data-area-name) is located. The data area is locked while the data is being retrieved.

The parameter *dtaptr* is a pointer to the storage that receives the retrieved copy of the data area. Only character data can be retrieved using QXXRTVDA.

Example that uses QXXRTVDA()

```
#include <stdio.h>
#include <xxdtaa.h>
#define DATA AREA LENGTH 30
#define START
                          6
#define LENGTH
                          7
int main(void)
 char uda area[DATA AREA LENGTH];
 /* Retrieve data from user-defined data area currently in MYLIB */
                                     ", "MYLIB
  DTAA NAME T dtaname = {"USRDDA
 /* Use the function to retrieve some data into uda area.
                                                                  */
 QXXRTVDA(dtaname, START, LENGTH, uda_area);
 /* Print the contents of the retrieved subset.
 printf("uda area contains %7.7s\n",uda area);
```

Related Information

• "QXXCHGDA() —Change Data Area" on page 218

QXXZTOD() -Convert Zoned Decimal to Double

Format

```
#include <xxcvt.h>
double QXXZTOD(unsigned char *zptr, int digits, int fraction);
```

Language Level: ILE C Extension

Description

The QXXZTOD function converts to a *double*, the zoned decimal number (with *digits* total digits, and *fraction* fractional digits) pointed to by *zptr*. The resulting *double* value is returned.

Example that uses QXXZTOD()

Related Information

- "QXXDTOP() Convert Double to Packed Decimal" on page 219
- "QXXDTOZ() —Convert Double to Zoned Decimal" on page 220
- "QXXITOP() —Convert Integer to Packed Decimal" on page 221
- "QXXITOZ() —Convert Integer to Zoned Decimal" on page 221
- "QXXPTOD() —Convert Packed Decimal to Double" on page 222
- "QXXPTOI() —Convert Packed Decimal to Integer" on page 223
- "QXXZTOI() —Convert Zoned Decimal to Integer"

QXXZTOI() —Convert Zoned Decimal to Integer

Format

```
#include <xxcvt.h>
int QXXZTOI(unsigned char *zptr, int digits, int fraction);
```

Language Level: ILE C Extension

Description

The QXXZTOI function converts to an *integer*, the zoned decimal number (with *digits* total digits, and *fraction* fractional digits) pointed to by *zptr*. The resulting integer is returned.

Example that uses QXXZTOI()

```
#include <xxcvt.h>
#include <stdio.h>
int main(void)
 unsigned char zptr[] = "000111115";
 int digits = 9, fraction = 0, result;
 result = QXXZTOI(zptr, digits, fraction);
                      /* result = 111115 */
```

- "QXXDTOP() Convert Double to Packed Decimal" on page 219
- "QXXDTOZ() —Convert Double to Zoned Decimal" on page 220
- "QXXITOP() —Convert Integer to Packed Decimal" on page 221
- "QXXITOZ() —Convert Integer to Zoned Decimal" on page 221
- "QXXPTOD() —Convert Packed Decimal to Double" on page 222
- "QXXPTOI() —Convert Packed Decimal to Integer" on page 223
- "QXXZTOD() —Convert Zoned Decimal to Double" on page 224

raise() — Send Signal

Format

```
#include <signal.h>
int raise(int sig);
```

Language Level: ANSI

Description

The raise() functions sends the signal *sig* to the running program. If compiled with SYSIFCOPT(*ASYNCSIGNAL) on the compilation command, this function uses asynchronous signals. The asynchronous version of this function throws a signal to the process or thread.

Return Value

The raise() functions returns 0 if successful, nonzero if unsuccessful.

Example that uses raise()

This example establishes a signal handler called sig_hand for the signal SIGUSR1. The signal handler is called whenever the SIGUSR1 signal is raised and will ignore the first nine occurrences of the signal. On the tenth raised signal, it exits the program with an error code of 10. Note that the signal handler must be reestablished each time it is called.

```
#include <signal.h>
#include <stdio.h>
void sig hand(int); /* declaration of sig hand() as a function */
int main(void)
   signal(SIGUSR1, sig hand); /* set up handler for SIGUSR1 */
   raise(SIGUSR1); /* signal SIGUSR1 is raised */
```

```
/* sig_hand() is called */

void sig_hand(int sig)
{
   static int count = 0; /* initialized only once */

   count++;
   if (count == 10) /* ignore the first 9 occurrences of this signal */
        exit(10);
   else
        signal(SIGUSR1, sig_hand); /* set up the handler again */
}
/* This is a program fragment and not a complete program */
```

- "signal() Handle Interrupt Signals" on page 313
- "Signal Handling Action Definitions" on page 456
- "<signal.h>" on page 14
- Signal APIs in the iSeries Information Center.
- POSIX thread API concepts in the iSeries Information Center.

rand(), rand_r() — Generate Random Number

Format

```
#include <stdlib.h>
int rand(void);
int rand r(unsigned int *seed);
```

Language Level: ANSI

Thread Safe: rand() is not thread safe, but rand_r() is.

Description

The rand() function generates a pseudo-random integer in the range 0 to RAND_MAX (macro defined in <stdlib.h>). Use the srand() function before calling rand() to set a starting point for the random number generator. If you do not call the srand() function first, the default seed is 1.

Note: The rand r() function is the restartable version of rand().

Return Value

The rand() function returns a pseudo-random number.

Example that uses rand()

This example prints the first 10 random numbers generated.

```
#include <stdlib.h>
#include <stdio.h>
int main(void)
   int x;
  for (x = 1; x \le 10; x++)
     printf("iteration %d, rand=%d\n", x, rand());
/****************** Output should be similar to: ********
iteration 1, rand=16838
iteration 2, rand=5758
iteration 3, rand=10113
iteration 4, rand=17515
iteration 5, rand=31051
iteration 6, rand=5627
iteration 7, rand=23010
iteration 8, rand=7419
iteration 9, rand=16212
iteration 10, rand=4086
```

- "srand() Set Seed for rand() Function" on page 319
- "<stdlib.h>" on page 16

_Racquire() —Acquire a Program Device

Format

```
#include <recio.h>
int _Racquire(_RFILE *fp, char *dev);
```

Language Level: ILE C Extension

Thread Safe: NO.

Description

The Racquire() function acquires the program device specified by the *dev* parameter and associates it with the file specified by fp. The dev parameter is a null-ended C string. The program device name must be specified in uppercase. The program device must be defined to the file. This function is valid for display and ICF files.

Return Value

The _Racquire() function returns 1 if it is successful or zero if it is unsuccessful. The value of errno may be set to EIOERROR (a non-recoverable I/O error occurred) or EIORECERR (a recoverable I/O error occurred).

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Racquire()

```
#include <stdio.h>
#include <recio.h>
#include <string.h>
#include <stdlib.h>
int main(void)
    RFILE
              *fp;
   _RIOFB_T *rfb;
   /* Open the device file.
                                                                     */
   if ((fp = Ropen ( "MYLIB/T1677RD2", "ar+" )) == NULL )
       printf ( "Could not open file\n" );
       exit ( 1 );
   Racquire ( fp,"DEVICE1" );
                                  /* Acquire another program device. */
                                  /* Replace with actual device name.*/
   Rformat ( fp,"FORMAT1" );
                                  /* Set the record format for the
                                  /* display file.
                                                                     */
   rfb = Rwrite (fp, "", 0); /* Set up the display.
                                                                     */
   /* Do some processing...
                                                                     */
   _Rclose (fp);
```

• "_Rrelease() — Release a Program Device" on page 283

_Rclose() —Close a File

|

Format

```
#include <recio.h>
int _Rclose(_RFILE *fp);
```

Language Level: ILE C Extension

Description

The _Rclose() function closes the file specified by *fp*. Before this file is closed, all buffers associated with it are flushed and all buffers reserved for it are released. The file is closed even if an exception occurs. The _Rclose() function applies to all types of files.

Note: Closing a file more than once in a multi-threaded environment will cause undefined behavior.

Return Value

The _Rclose() function returns zero if the file is closed successfully, or EOF if the close operation failed or the file was already closed. The file is closed even if an exception occurs, and zero is returned.

The value of errno may be set to:

Value	Meaning
ENOTOPEN	The file is not open.
EIOERROR	A non-recoverable I/O error occurred.
EIORECERR	A recoverable I/O error occurred.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses Rclose()

```
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
int main(void)
    _RFILE
               *fp;
   /* Open the file for processing in arrival sequence.
   if (( fp = _Ropen ( "MYLIB/T1677RD1", "rr+, arrseq=Y" )) == NULL )
       printf ( "Open failed\n" );
       exit (1);
   else
    /* Do some processing */;
    Rclose (fp);
```

Related Information

• "_Ropen() — Open a Record File for I/O Operations" on page 257

_Rcommit() —Commit Current Record

Format

```
#include <recio.h>
int Rcommit(char *cmtid);
```

Language Level: ILE C Extension

Thread Safe: NO.

Description

The _Rcommit() function completes the current transaction for the job that calls it and establishes a new commitment boundary. All changes made since the last commitment boundary are made permanent. Any file or resource that is open under commitment control in the job is affected.

The cmtid parameter is a null-ended C string used to identify the group of changes associated with a commitment boundary. It cannot be longer than 4000 bytes.

The Rcommit() function applies to database and DDM files.

Return Value

The _Rcommit() function returns 1 if the operation is successful or zero if the operation is unsuccessful. The value of errno may be set to EIOERROR (a non-recoverable I/O error occurred) or EIORECERR (a recoverable I/O error occurred).

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rcommit()

```
#include <stdio.h>
#include <recio.h>
#include <stdlib.h>
#include <string.h>
int main(void)
            buf[40];
 char
 int
            rc = 1;
 _RFILE
            *purf;
  RFILE
            *dailyf;
 /* Open purchase display file and daily transaction file
 if ( ( purf = _Ropen ( "MYLIB/T1677RD3", "ar+,indicators=y" )) == NULL )
  {
     printf ( "Display file did not open.\n" );
     exit (1);
 if ( ( dailyf = Ropen ( "MYLIB/T1677RDA", "wr,commit=y") ) == NULL )
     printf ( "Daily transaction file did not open.\n" );
     exit (2);
 /* Select purchase record format */
  Rformat ( purf, "PURCHASE" );
  /* Invite user to enter a purchase transaction.
 /* The Rwrite function writes the purchase display.
  _Rwrite ( purf, "", 0 );
  _Rreadn ( purf, buf, sizeof(buf), __DFT );
 /* Update daily transaction file
 rc = (( _Rwrite ( dailyf, buf, sizeof(buf) ))->num_bytes );
  /* If the databases were updated, then commit the transaction.
  /* Otherwise, rollback the transaction and indicate to the
  /* user that an error has occurred and end the application.
 if ( rc )
   {
        Rcommit ( "Transaction complete" );
   }
 else
   {
        Rrollbck ( );
       _Rformat ( purf, "ERROR" );
    }
  Rclose ( purf );
  Rclose (dailyf);
```

Related Information

_Rdelete() —Delete a Record

Format

```
#include <recio.h>
_RIOFB_T *_Rdelete(_RFILE *fp);
```

Language Level: ILE C Extension

Description

The _Rdelete() function deletes the record that is currently locked for update in the file specified by fp. After the delete operation, the record is not locked. The file must be open for update.

A record is locked for update by reading or locating to it unless __NO_LOCK is specified on the read or locate option. If the __NO_POSITION option is specified on the locate operation that locked the record, the record deleted may not be the record that the file is currently positioned to.

This function is valid for database and DDM files.

Return Value

The Rdelete() function returns a pointer to the _RIOFB_T structure associated with fp. If the operation is successful, the num_bytes field contains 1. If the operation is unsuccessful, the num_bytes field contains zero.

The value of errno may be set to:

Value Meaning

ENOTDLT

The file is not open for delete operations.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses Rdelete()

```
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
int main(void)
     RFILE
                *fp;
    XXOPFB_T
              *opfb;
    /* Open the file for processing in arrival sequence.
    if (( fp = Ropen ( "MYLIB/T1677RD1", "rr+, arrseq=Y" )) == NULL )
        printf ( "Open failed\n" );
       exit ( 1 );
    /* Get the library and file names of the file opened.
                                                                      */
   opfb = _Ropnfbk ( fp );
   printf ( "Library: %10.10s\nFile:
                                         %10.10s\n",
              opfb->library_name,
              opfb->file name);
    /* Get the first record.
                                                                      */
    _Rreadf ( fp, NULL, 20, __DFT );
   printf ("First record: \frac{10.10s}{n}, *(fp->in buf));
    /* Delete the first record.
                                                                      */
    _Rdelete ( fp );
    _Rclose (fp);
```

• "_Rrlslck() — Release a Record Lock" on page 285

_Rdevatr() —Get Device Attributes

Format

```
#include <recio.h>
#include <xxfdbk.h>
XXDEV ATR T * Rdevatr( RFILE *fp, char *dev);
```

Language Level: ILE C Extension

Thread Safe: NO.

Description

The _Rdevatr() function returns a pointer to a copy of the device attributes feedback area for the file pointed to by *fp*, and the device specified by *dev*.

The *dev* parameter is a null-ended C string. The device name must be specified in uppercase.

The _Rdevatr() function is valid for display and ICF files.

Return Value

The _Rdevatr() function returns NULL if an error occurs.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rdevatr()

```
#include <stdio.h>
#include <recio.h>
#include <string.h>
#include <stdlib.h>
int main(int argc, char ** argv)
    _RFILE *fp; /* File pointer
   _RIOFB_T *rfb; /*Pointer to the file's feedback structure
   _XXIOFB_T *iofb; /* Pointer to the file's feedback area
    _XXDEV_ATR_T *dv_atr; /* Pointer to a copy of the file's device
                          /* attributes feedback area
    /* Open the device file.
    if (( fp = _Ropen ( "MYLIB/T1677RD2", "ar+" )) == NULL )
       printf ( "Could not open file\n" );
       exit (1);
    dv atr = Rdevatr (fp, argv[1]);
    if (dv atr == NULL)
      printf("Error occurred getting device attributes for %s.\n",
    Rclose (fp);
```

Related Information

- "_Racquire() —Acquire a Program Device" on page 228
- "_Rrelease() Release a Program Device" on page 283

realloc() — Change Reserved Storage Block Size

Format

```
#include <stdlib.h>
void *realloc(void *ptr, size t size);
```

Language Level: ANSI

Thread Safe: YES

Description

The realloc() function changes the size of a previously reserved storage block. The ptr argument points to the beginning of the block. The size argument gives the new size of the block, in bytes. The contents of the block are unchanged up to the shorter of the new and old sizes.

If the ptr is NULL, realloc() reserves a block of storage of size bytes. It does not necessarily give all bits of each element an initial value of 0.

If size is 0 and the ptr is not NULL, realloc() frees the storage allocated to ptr and returns NULL

Return Value

The realloc() function returns a pointer to the reallocated storage block. The storage location of the block may be moved by the realloc() function. Thus, the *ptr* argument to the realloc() function is not necessarily the same as the return value.

If *size* is 0, the realloc() function returns NULL. If there is not enough storage to expand the block to the given size, the original block is unchanged and the realloc() function returns NULL.

The storage to which the return value points is aligned for storage of any type of object.

Note: By default, the ILE C memory management functions malloc(), calloc() and realloc() obtain storage from the ILE Cfast heap, which initially is 64944 bytes in size. This can be increased up to about 16 megabytes by changing the value of the _HEAP_SIZE macro (defined in the <stdio.h> include file).

To change the size of the fast heap, you must do so before using malloc(), calloc() or realloc() in your program.

The use of the fast heap can be bypassed by setting the _HEAP_SIZE macro to the value of _NO_DEFAULT_HEAP, also defined in the <stdlib.h> include file. See "Chapter 1. Include Files" on page 3 for a description of <stdlib.h>. Storage allocated from the fast heap by realloc() cannot be freed or reallocated with the ILE bindable APIs CEEFRST and CEECZST. Storage initially allocated with the ILE bindable API CEEGTST can be reallocated with the realloc() function.

To use **Teraspace** storage instead of heap storage without changing the C source code, specify the TERASPACE(*YES *TSIFC) parameter on the CRTCMOD compiler command. This maps the realloc() library function to _C_TS_realloc(), its Teraspace storage counterpart. The maximum amount of Teraspace storage that can be allocated by each call to _C_TS_realloc() is 2GB - 240, or 214743408 bytes. For additional information about Teraspace, see the *ILE Concepts* manual.

Example that uses realloc()

This example allocates storage for the prompted size of array and then uses realloc() to reallocate the block to hold the new size of the array. The contents of the array are printed after each allocation.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
 long * array;
                  /* start of the array */
 long * ptr;
                 /* pointer to array */
 int i;
                 /* index variable */
 int num1, num2; /* number of entries of the array */
 void print_array( long *ptr_array, int size);
 printf( "Enter the size of the array\n" );
 scanf("%i", &num1);
 /* allocate num1 entries using malloc() */
 if ( (array = (long *) malloc( num1 * sizeof( long ))) != NULL )
    for (ptr = array, i = 0; i < num1; ++i) /* assign values */
       *ptr++ = i;
    print array( array, num1 );
    printf("\n");
 else { /* malloc error */
   perror( "Out of storage" );
   abort();
 /* Change the size of the array ... */
 printf( "Enter the size of the new array\n" );
 scanf( "%i", &num2);
 if ( (array = (long *) realloc( array, num2* sizeof( long ))) != NULL )
    for ( ptr = array + num1, i = num1; i <= num2; ++i )
       *ptr++ = i + 2000; /* assign values to new elements */
    print_array( array, num2 );
 else { /* realloc error */
   perror( "Out of storage" );
   abort();
void print array( long * ptr array, int size )
 int i;
 long * index = ptr array;
 printf("The array of size %d is:\n", size);
 for ( i = 0; i < size; ++i )
                               /* print the array out
                                                                 */
   printf( " array[%i] = %li\n", i, ptr_array[i]);
/**** If the initial value entered is 2 and the second value entered
     is 4, then the expected output is:
Enter the size of the array
The array of size 2 is:
 array[0] = 0
 array[1] = 1
Enter the size of the new array
The array of size 4 is:
 array[0] = 0
 array[ 1 ] = 1
 array[2] = 2002
 array[3] = 2003
                                                                 */
```

- "calloc() Reserve and Initialize Storage" on page 56
- "free() Release Storage Blocks" on page 109
- "malloc() Reserve Storage Block" on page 170

regcomp() — Compile Regular Expression

Format

I

#include <regex.h>
int regcomp(regex_t *preg, const char *pattern, int cflags);

Language Level: XPG4

Thread Safe: NO.

Description

The regcomp() function compiles the source regular expression pointed to by *pattern* into an executable version and stores it in the location pointed to by *preg*. You can then use the regexec()function to compare the regular expression to other strings.

The cflags flag defines the attributes of the compilation process:

errcode	Description String
REG_EXTENDED	Support extended regular expressions.
REG_NEWLINE	Treat new-line character as a special end-of-line character; it then establishes the line boundaries matched by the] and \$ patterns, and can only be matched within a string explicitly using \n. (If you omit this flag, the new-line character is treated like any other character.)
REG_ICASE	Ignore case in match.
REG_NOSUB	Ignore the number of subexpressions specified in <i>pattern</i> . When you compare a string to the compiled pattern (using regexec()), the string must match the entire pattern. The regexec()function then returns a value that indicates only if a match was found; it does not indicate at what point in the string the match begins, or what the matching string is.

Regular expressions are a context-independent syntax that can represent a wide variety of character sets and character set orderings, which can be interpreted differently depending on the current locale. The functions regcomp(), regerror(), regexec(), and regfree() use regular expressions in a similar way to the UNIX® awk, ed, grep, and egrep commands.

Note: This function is accessible only when LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

Return Value

If the regcomp() function is successful, it returns 0. Otherwise, it returns an error code that you can use in a call to the regerror() function, and the content of *preg* is undefined.

Example that uses regcomp()

```
#include <regex.h>
#include <stdio.h>
#include <stdlib.h>
int main(void)
  regex t
  char
            *string = "a very simple simple simple string";
            *pattern = "\\(sim[a-z]le\\) \\1";
  char
  int
            rc;
            nmatch = 2;
  size t
  regmatch_t pmatch[2];
  if (0 != (rc = regcomp(&preg, pattern, 0))) {
     printf("regcomp() failed, returning nonzero (%d)\n", rc);
     exit(EXIT FAILURE);
  if (0 != (rc = regexec(&preg, string, nmatch, pmatch, 0))) {
     printf("Failed to match '%s' with '%s', returning %d.\n",
     string, pattern, rc);
  else {
     printf("With the whole expression, "
            "a matched substring \"%.*s\" is found at position %d to %d.\n",
            pmatch[@].rm\_eo - pmatch[@].rm\_so, \&string[pmatch[@].rm\_so],\\
            pmatch[0].rm_so, pmatch[0].rm_eo - 1);
     printf("With the sub-expression,
            "a matched substring \"\%.*s\" is found at position %d to %d.\n",
            pmatch[1].rm eo - pmatch[1].rm so, "string[pmatch[1].rm so],
            pmatch[1].rm so, pmatch[1].rm eo - 1);
  regfree(&preg);
  return 0;
  /***********************************
     The output should be similar to:
     With the whole expression, a matched substring "simple simple" is found
     at position 7 to 19.
     With the sub-expression, a matched substring "simple" is found
     at position 7 to 12.
```

Related Information

- "regerror() Return Error Message for Regular Expression"
- "regexec() Execute Compiled Regular Expression" on page 240
- "regfree() Free Memory for Regular Expression" on page 242
- "<regex.h>" on page 13

regerror() — Return Error Message for Regular Expression

Format

```
#include <regex.h>
size t regerror(int errcode, const regex t *preg,
               char *errbuf, size t errbuf size);
```

Language Level: XPG4

Description

The regerror() function finds the description for the error code *errcode* for the regular expression *preg*. The description for *errcode* is assigned to *errbuf*. The *errbuf_size* value specifies the maximum message size that can be stored (the size of errbuf). The description strings for *errcode* are:

errcode	Description String
REG_NOMATCH	regexec() failed to find a match.
REG_BADPAT	Invalid regular expression.
REG_ECOLLATE	Invalid collating element referenced.
REG_ECTYPE	Invalid character class type referenced.
REG_EESCAPE	Last character in regular expression is a \.
REG_ESUBREG	Number in \digit invalid, or error.
REG_EBRACK	[] imbalance.
REG_EPAREN	\(\) or () imbalance.
REG_EBRACE	\{ \} imbalance.
REG_BADBR	Expression between \{ and \} is invalid.
REG_ERANGE	Invalid endpoint in range expression.
REG_ESPACE	Out of memory.
REG_BADRPT	?, *, or + not preceded by valid regular expression.
REG_ECHAR	Invalid multibyte character.
REG_EBOL	anchor not at beginning of regular expression.
REG_EEOL	\$ anchor not at end of regular expression.
REG_ECOMP	Unknown error occurred during regcomp() call.
REG_EEXEC	Unknown error occurred during regexec() call.

Note: This function is accessible only when LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

Return Value

The regerror() returns the size of the buffer needed to hold the string that describes the error condition.

Example that uses regerror()

This example compiles an invalid regular expression, and prints an error message using the regerror() function.

```
#include <regex.htm>
#include <stdio.h>
#include <stdlib.h>
int main(void)
  regex t preg;
         *pattern = "a[missing.bracket";
  char
  int
         rc;
  char
         buffer[100];
  if (0 != (rc = regcomp(&preg, pattern, REG EXTENDED))) {
     regerror(rc, &preg, buffer, 100);
     printf("regcomp() failed with '%s'\n", buffer);
     exit(EXIT FAILURE);
  return 0;
/*********************
     The output should be similar to:
     regcomp() failed with '[] imbalance.'
  *************************************
```

- "regcomp() Compile Regular Expression" on page 237
- "regexec() Execute Compiled Regular Expression"
- "regfree() Free Memory for Regular Expression" on page 242
- "<regex.h>" on page 13

regexec() — Execute Compiled Regular Expression

Format

```
#include <regex.h>
int regexec(const regex t *preg, const char *string,
            size t nmatch, regmatch t *pmatch, int eflags);
```

Language Level: XPG4

Thread Safe: NO.

Description

The regexec() function compares the null-ended string against the compiled regular expression preg to find a match between the two.

The *nmatch* value is the number of substrings in *string* that the regexec() function should try to match with subexpressions in preg. The array you supply for pmatch must have at least *nmatch* elements.

The regexec() function fills in the elements of the array *pmatch* with offsets of the substrings in string that correspond to the parenthesized subexpressions of the original pattern given to the regcomp() function to create preg. The zeroth element of the array corresponds to the entire pattern. If there are more than *nmatch* subexpressions, only the first *nmatch* - 1 are stored. If *nmatch* is 0, or if the REG_NOSUB flag was set when preg was created with the regcomp() function, the regexec() function ignores the *pmatch* argument.

The eflags *flag* defines customizable behavior of the regexec() function:

errcode	Description String
REG_NOTBOL	Indicates that the first character of <i>string</i> is not the beginning of line.
REG_NOTEOL	Indicates that the first character of <i>string</i> is not the end of line.

When a basic or extended regular expression is matched, any given parenthesized subexpression of the original pattern could participate in the match of several different substrings of *string*. The following rules determine which substrings are reported in *pmatch*:

- 1. If subexpression *i* in a regular expression is not contained within another subexpression, and it participated in the match several times, then the byte offsets in *pmatch[i]* will delimit the last such match.
- 2. If subexpression *i* is not contained within another subexpression, and it did not participate in an otherwise successful match, the byte offsets in *pmatch[i]* will be -1. A subexpression does not participate in the match when any of following conditions are true:
 - * or \{ \} appears immediately after the subexpression in a basic regular expression.
 - *, ?, or { } appears immediately after the subexpression in an extended regular expression, and the subexpression did not match (matched 0 times).
 - I is used in an extended regular expression to select this subexpression or another, and the other subexpression matched.
- 3. If subexpression *i* is contained within another subexpression *j*, and *i* is not contained within any other subexpression that is contained within *j*, and a match of subexpression *j* is reported in *pmatch[j]*, then the match or non-match of subexpression *i* reported in *pmatch[i]* will be as described in 1. and 2. above, but within the substring reported in *pmatch[j]* rather than the whole string.
- 4. If subexpression *i* is contained in subexpression *j*, and the byte offsets in *pmatch*[*i*] are -1, then the offsets in *pmatch*[*i*] also will be -1. \
- 5. If subexpression *i* matched a zero-length string, then both byte offsets in *pmatch[i]* will be the byte offset of the character or null terminator immediately following the zero-length string.

If the REG_NOSUB flag was set when *preg* was created by the regcomp() function, the contents of *pmatch* are unspecified. If the REG_NEWLINE flag was set when *preg* was created, new-line characters are allowed in string.

Return Value

If a match is found, the regexec() function returns 0. If no match is found, the regexec() function returns REG_NOMATCH. Otherwise, it returns a nonzero value indicating an error. A nonzero return value can be used in a call to the regerror() function.

Example that uses regexec()

```
#include <regex.h>
#include <stdio.h>
#include <stdlib.h>
int main(void)
  regex t
            *string = "a very simple simple string";
  char
            *pattern = "\\(sim[a-z]le\\) \\1";
  char
  int
            rc:
  size t
            nmatch = 2;
  regmatch t pmatch[2];
  if (0 != (rc = regcomp(&preg, pattern, 0))) {
     printf("regcomp() failed, returning nonzero (%d)\n", rc);
     exit(EXIT_FAILURE);
  if (0 != (rc = regexec(&preg, string, nmatch, pmatch, 0))) {
   printf("Failed to match '%s' with '%s',returning %d.\n",
     string, pattern, rc);
  else {
     printf("With the whole expression, "
            "a matched substring \"%.*s\" is found at position %d to %d.\n",
            pmatch[0].rm_eo - pmatch[0].rm_so, &string[pmatch[0].rm_so],
            pmatch[0].rm_so, pmatch[0].rm_eo - 1);
     printf("With the sub-expression,
            "a matched substring \"%.*s\" is found at position %d to %d.\n",
            pmatch[1].rm_eo - pmatch[1].rm_so, &string[pmatch[1].rm_so],
           pmatch[1].rm_so, pmatch[1].rm_eo - 1);
  regfree(&preg);
  return 0;
The output should be similar to:
     With the whole expression, a matched substring "simple simple" is found
     at position 7 to 19.
     With the sub-expression, a matched substring "simple" is found
     at position 7 to 12.
  }
```

- "regcomp() Compile Regular Expression" on page 237
- "regerror() Return Error Message for Regular Expression" on page 238
- "regfree() Free Memory for Regular Expression"
- "<regex.h>" on page 13

regfree() — Free Memory for Regular Expression

Format

```
#include <regex.h>
void regfree(regex_t *preg);
```

Language Level: XPG4

Description

The regfree() function frees any memory that was allocated by the regcomp()function to implement the regular expression *preg*. After the call to the regfree() function, the expression that is defined by *preg* is no longer a compiled regular or extended expression.

Note: This function is accessible only when LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

Return Value

There is no return value.

Example that uses regfree()

This example compiles an extended regular expression.

```
#include <regex.h>
#include <stdio.h>
#include <stdlib.h>
int main(void)
  regex_t preg;
        *pattern = ".*(simple).*";
  char
  int
  if (0 != (rc = regcomp(&preg, pattern, REG_EXTENDED))) {
     printf("regcomp() failed, returning nonzero (%d)\n", rc);
     exit(EXIT_FAILURE);
  regfree(&preg);
  printf("regcomp() is successful.\n");
  return 0;
/**********************************
     The output should be similar to:
     regcomp() is successful.
```

Related Information

- "regcomp() Compile Regular Expression" on page 237
- "regerror() Return Error Message for Regular Expression" on page 238
- "regexec() Execute Compiled Regular Expression" on page 240
- "<regex.h>" on page 13

remove() — Delete File

Format

```
#include <stdio.h>
int remove(const char *filename);
```

Language Level: ANSI

Description

The remove() function deletes the file specified by *filename*. If the filename contains the member name, the member is removed or the file is deleted.

Note: You cannot remove a nonexistent file or a file that is open.

Return Value

The remove() function returns 0 if it successfully deletes the file. A nonzero return value indicates an error.

Example that uses remove()

When you call this example with a file name, the program attempts to remove that file. It issues a message if an error occurs.

```
#include <stdio.h>
int main(int argc, char ** argv)
 if ( argc != 2 )
   printf( "Usage: %s fn\n", argv[0] );
    if ( remove( argv[1] ) != 0 )
     perror( "Could not remove file" );
```

Related Information

- "fopen() Open Files" on page 92
- "rename() Rename File"
- "<stdio.h>" on page 15

rename() — Rename File

Format

```
#include <stdio.h>
int rename(const char *oldname, const char *newname);
```

Language Level: ANSI

Description

The rename() function renames the file specified by oldname to the name given by newname. The oldname pointer must specify the name of an existing file. The newname pointer must not specify the name of an existing file. You cannot rename a file with the name of an existing file. You also cannot rename an open file.

The file formats that can be used to satisfy the new name depend on the format of the old name. The following table shows the valid file formats that can be used to specify the old file name and the corresponding valid file formats for the new name.

If the format for both new name and old name is lib/file(member), then the file cannot change. If the file name changes, rename will not work. For example, the following is not valid: lib/file1(member1) lib/file2(member1).

Old Name	New Name
lib/file(member)	lib/file(member), lib/file, file, file(member)
lib/file	lib/file, file
file	lib/file, file
file(member)	lib/file(member), lib/file, file, file(member)

Return Value

The rename() function returns 0 if successful. On an error, it returns a nonzero value.

Example that uses rename()

This example takes two file names as input and uses rename() to change the file name from the first name to the second name.

```
#include <stdio.h>
int main(int argc, char ** argv )
{
  if ( argc != 3 )
    printf( "Usage: %s old_fn new_fn\n", argv[0] );
  else if ( rename( argv[1], argv[2] ) != 0 )
    perror ( "Could not rename file" );
}
```

Related Information

- "fopen() Open Files" on page 92
- "remove() Delete File" on page 243
- "<stdio.h>" on page 15

rewind() — Adjust Current File Position

Format

```
#include <stdio.h>
void rewind(FILE *stream);
```

Language Level: ANSI

Description

The rewind() function repositions the file pointer associated with *stream* to the beginning of the file. A call to the rewind() function is the same as:

```
(void)fseek(stream, OL, SEEK_SET);
```

except that the rewind() function also clears the error indicator for the stream.

The rewind() function is not supported for files opened with type=record.

Return Value

There is no return value.

The value of errno may be set to:

Value Meaning

EBADF

The file pointer or descriptor is not valid.

ENODEV

Operation attempted on a wrong device.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

Example that uses rewind()

This example first opens a file myfile for input and output. It writes integers to the file, uses rewind() to reposition the file pointer to the beginning of the file, and then reads in the data.

```
#include <stdio.h>
FILE *stream;
int data1, data2, data3, data4;
int main(void)
  data1 = 1; data2 = -37;
      /* Place data in the file */
  stream = fopen("mylib/myfile", "w+");
   fprintf(stream, "%d %d\n", data1, data2);
      /* Now read the data file */
  rewind(stream);
  fscanf(stream, "%d", &data3);
fscanf(stream, "%d", &data4);
  printf("The values read back in are: %d and %d\n",
       data3, data4);
/******* Output should be similar to: ********
The values read back in are: 1 and -37
```

Related Information

- "fgetpos() Get File Position" on page 84
- "fseek() fseeko() Reposition File Position" on page 114
- "fsetpos() Set File Position" on page 117
- "ftell() ftello() Get Current Position" on page 118
- "<stdio.h>" on page 15

_Rfeod() —Force the End-of-Data

Format

```
#include <recio.h>
int Rfeod( RFILE *fp);
```

Language Level: ILE C Extension

Description

The _Rfeod() function forces an end-of-data condition for a device or member associated with the file specified by *fp*. Any outstanding updates, deletes or writes that the system is buffering will be forced to nonvolatile storage. If a database file is open for input, any outstanding locks will be released.

The _Rfeod() function positions the file to *END unless the file is open for multi-member processing and the current member is not the last member in the file. If multi-member processing is in effect and the current member is not the last member in the file, _Rfeod() will open the next member of the file and position it to *START.

The _Rfeod() function is valid for all types of files.

Return Value

The _Rfeod() function returns 1 if multi-member processing is taking place and the next member has been opened. EOF is returned if the file is positioned to *END. If the operation is unsuccessful, zero is returned. The value of errno may be set to EIOERROR (a non-recoverable error occurred) or EIORECERR (a recoverable I/O error occurred). See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses Rfeod()

```
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
int main(void)
    RFILE
            new_purchase[21] = "PEAR
   char
                                           1002022244";
    /* Open the file for processing in keyed sequence.
                                                                     */
    if ( (in = Ropen("MYLIB/T1677RD4", "rr+, arrseq=N")) == NULL )
       printf("Open failed\n");
        exit(1);
    };
    /* Update the first record in the keyed sequence.
                                                                     */
    Rlocate(in, NULL, 0,
                           FIRST);
    _Rupdate(in, new_purchase, 20);
    /* Force the end of data.
                                                                     */
    Rfeod(in);
```

Related Information

- "_Racquire() —Acquire a Program Device" on page 228
- "_Rfeov() —Force the End-of-File"

Rfeov() —Force the End-of-File

Format

```
#include <recio.h>
int Rfeov( RFILE *fp);
```

Language Level: ILE C Extension

Description

The Rfeov() function forces an end-of-volume condition for a tape file that is associated with the file that is specified by fp. The Rfeov() function positions the file to the next volume of the file. If the file is open for output, the output buffers will be flushed.

The _Rfeov() function is valid for tape files.

Return Value

The _Rfeov() function returns 1 if the file has moved from one volume to the next. It will return EOF if it is called while processing the last volume of the file. It will return zero if the operation is unsuccessful. The value of errno may be set to EIOERROR (a non-recoverable error occurred) or EIORECERR (a recoverable I/O error occurred). See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rfeov()

```
#include <stdio.h>
#include <recio.h>
#include <stdlib.h>
int main(void)
    RFILE *tape;
    RFILE *fp;
    char buf[92];
    int
         i, feov2;
                                                                 */
   /* Open source physical file containing C source.
    if (( fp = _Ropen ( "QCSRC(T1677SRC)", "rr blkrcd=y" )) == NULL )
        printf ( "could not open C source file\n" );
        exit (1);
    }
    /* Open tape file to receive C source statements
    if (( tape = Ropen ( "T1677TPF", "wr lrecl=92 blkrcd=y" )) == NULL )
       printf ( "could not open tape file\n" );
        exit (2);
    /* Read the C source statements, find their sizes
                                                                 */
    /* and add them to the tape file.
   while (( _Rreadn ( fp, buf, sizeof(buf), __DFT )) -> num_bytes != EOF
)
        for ( i = sizeof(buf) - 1; buf[i] == ' ' && i > 12;
        i = (i == 12) ? 80 : (1-12);
       memmove( buf, buf+12, i );
        _Rwrite ( tape, buf, i );
    feov2 = _Rfeov (fp);
    Rclose (fp);
    Rclose ( tape );
```

- "_Racquire() —Acquire a Program Device" on page 228
- "_Rfeod() —Force the End-of-Data" on page 246

_Rformat() —Set the Record Format Name

Format

```
#include <recio.h>
void _Rformat(_RFILE *fp, char *fmt);
```

Language Level: ILE C Extension

Description

The _Rformat() function sets the record format to *fmt* for the file specified by *fp*.

The _Rformat() function is valid for multi-format logical database, DDM files, display, ICF and printer files.

The fmt parameter is a null-ended C string. The fmt parameter must be in uppercase.

Return Value

The Rformat() function returns void. See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rformat()

This example shows how Rformat() is used.

```
#include <stdio.h>
#include <recio.h>
#include <stdlib.h>
#include <string.h>
int main(void)
 char
            buf[40];
 int
            rc = 1;
  RFILE
            *purf;
 RFILE
            *dailyf;
  /* Open purchase display file and daily transaction file
 if ( ( purf = _Ropen ( "MYLIB/T1677RD3", "ar+,indicators=y" )) == NULL )
     printf ( "Display file did not open.\n" );
     exit (1);
 if ( ( dailyf = Ropen ( "MYLIB/T1677RDA", "wr,commit=y") ) == NULL )
     printf ( "Daily transaction file did not open.\n" );
     exit (2);
 /* Select purchase record format */
 Rformat ( purf, "PURCHASE" );
  /* Invite user to enter a purchase transaction.
 /* The Rwrite function writes the purchase display.
  _Rwrite ( purf, "", 0 );
 _Rreadn ( purf, buf, sizeof(buf), __DFT );
 /* Update daily transaction file
                                                                    */
 rc = (( _Rwrite ( dailyf, buf, sizeof(buf) ))->num_bytes );
 /* If the databases were updated, then commit the transaction.
                                                                    */
 /* Otherwise, rollback the transaction and indicate to the
  /* user that an error has occurred and end the application.
 if (rc)
        _Rcommit ( "Transaction complete" );
 else
    {
        _Rrollbck ();
       _Rformat ( purf, "ERROR" );
  Rclose ( purf );
  Rclose (dailyf);
```

• "_Ropen() — Open a Record File for I/O Operations" on page 257

_Rindara() —Set Separate Indicator Area

Format

#include <recio.h>
void _Rindara(_RFILE *fp, char *indic_buf);

Language Level: ILE C Extension

Thread Safe: NO.

Description

The _Rindara() function registers *indic_buf* as the separate indicator area to be used by the file specified by *fp*. The file must be opened with the keyword indicators=Y on the _Ropen() function. The DDS for the file should specify also that a separate indicator area is to be used. It is generally best to initialize a separate indicator area explicitly with '0' (character) in each byte.

The _Rindara() function is valid for display, ICF, and printer files.

Return Value

The _Rindara() function returns void. See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rindara()

```
#include <stdio.h>
#include <recio.h>
#include <stdlib.h>
#include <string.h>
#define PF03
#define IND OFF '0'
#define IND_ON '1'
int main(void)
 char
            buf[40];
 int
            rc = 1;
  SYSindara ind area;
  RFILE
            *purf;
            *dailyf;
 /* Open purchase display file and daily transaction file
 if ( ( purf = _Ropen ( "MYLIB/T1677RD3", "ar+,indicators=y" )) == NULL )
     printf ( "Display file did not open.\n" );
     exit ( 1 );
 if ( ( dailyf = Ropen ( "MYLIB/T1677RDA", "wr,commit=y") ) == NULL )
     printf ( "Daily transaction file did not open.\n" );
     exit (2);
 /* Associate separate indicator area with purchase file
                                                                    */
  _Rindara ( purf, ind_area );
 /* Select purchase record format */
  _Rformat ( purf, "PURCHASE" );
 7* Invite user to enter a purchase transaction.
 /* The _Rwrite function writes the purchase display.
  _Rwrite ( purf, "", 0 );
  Rreadn ( purf, buf, sizeof(buf), DFT );
 /* While user is entering transactions, update daily and
                                                                    */
  /* monthly transaction files.
 while ( rc && ind_area[PF03] == IND_OFF )
    rc = (( Rwrite ( dailyf, buf, sizeof(buf) ))->num bytes );
  /* If the databases were updated, then commit transaction
  /* otherwise, rollback the transaction and indicate to the
  /* user that an error has occurred and end the application.
   if ( rc )
        Rcommit ( "Transaction complete" );
    else
    {
        _Rrollbck ();
       _Rformat ( purf, "ERROR" );
    Rwrite ( purf, "", 0 );
    _Rreadn ( purf, buf, sizeof(buf), __DFT );
  _Rclose ( purf );
  Rclose (dailyf);
```

"_Ropen() — Open a Record File for I/O Operations" on page 257

_Riofbk() —Obtain I/O Feedback Information

Format

```
#include <recio.h>
#include <xxfdbk.h>
_XXIOFB_T *_Riofbk(_RFILE *fp);
```

Language Level: ILE C Extension

Description

The _Riofbk() function returns a pointer to a copy of the I/O feedback area for the file that is specified by *fp*.

The _Riofbk() function is valid for all types of files.

Return Value

The _Riofbk() function returns NULL if an error occurs. See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Riofbk()

```
#include <stdio.h>
#include <recio.h>
#include <string.h>
#include <stdlib.h>
typedef struct {
    char name[20];
    char address[25];
} format1;
typedef struct {
    char name[8];
    char password[10];
} format2:
typedef union {
    format1 fmt1;
    format2 fmt2;
} formats;
int main(void)
    RFILE *fp; /* File pointer
    RIOFB T *rfb; /*Pointer to the file's feedback structure
    XXIOFB T *iofb; /* Pointer to the file's feedback area
    formats buf, in buf, out buf; /* Buffers to hold data
                                                                         */
 /* Open the device file.
    if (( fp = Ropen ( "MYLIB/T1677RD2", "ar+" )) == NULL )
        printf ( "Could not open file\n" );
        exit (1);
    }
        Racquire ( fp,"DEVICE1" );
                                        /* Acquire another device. Replace
                                    /* with actual device name.
    Rformat ( fp,"FORMAT1" );
                                    /* Set the record format for the
                                                                         */
                                    /* display file.
                                                                         */
   rfb = _Rwrite ( fp, "", 0 );  /* Set up the display. */
_Rpgmdev ( fp, "DEVICE2" );  /* Change the default program device. */
                                 /* Replace with actual device name.
    _Rformat ( fp,"FORMAT2" );
                                 /* Set the record format for the
                                  /* display file.
                                                                         */
    rfb = Rwrite ( fp, "", 0 ); /* Set up the display.
                                                                         */
    rfb = _Rwriterd ( fp, &buf, sizeof(buf) );
    rfb = _Rwrread ( fp, &in_buf, sizeof(in_buf), &out_buf,
                      sizeof(out_buf ));
    _Rreadindv ( fp, &buf, sizeof(buf), _
                                           DFT );
                                   /* Read from the first device that */
```

```
/* enters data - device becomes
                                 /* default program device.
/* Determine which terminal responded first.
   iofb = _Riofbk ( fp );
   if ( !strncmp ( "FORMAT1 ", iofb -> rec_format, 10 ))
       Rrelease ( fp, "DEVICE1" );
   else
   {
      _Rrelease(fp, "DEVICE2");
/* Continue processing.
                                                                  */
   printf ( "Data displayed is %45.45s\n", &buf);
   Rclose (fp);
```

"_Ropnfbk() — Obtain Open Feedback Information" on page 261

_Rlocate() —Position a Record

Format

```
#include <recio.h>
RIOFB T * Rlocate( RFILE *fp, void *key, int klen rrn, int opts);
```

Language Level: ILE C Extension

Thread Safe: YES. However, if the file pointer is passed among threads, the I/O feedback area is shared among those threads.

Description

The Rlocate() function positions to the record in the file associated with fp and specified by the key, klen_rrn and opts parameters. The Rlocate() function locks the record specified by the key, klen_rrn and opts parameters unless __NO_LOCK is specified.

The Rlocate() function is valid for database and DDM files that are opened with the Ropen() function. The following are valid parameters of the Rlocate() function.

Points to a string containing the key fields to be used for positioning. key

klen_rrn

Specifies the length of the key that is used if positioning by key or the relative record number if positioning by relative record number.

opts Specifies positioning options to be used for the locate operation. The possible macros are:

__DFT

Default to __KEY_EQ and lock the record for update if the file is open for updating.

__END

Positions to just after the last record in a file. There is no record that is associated with this position.

_END_FRC

Positions to just after the last record in a file. All buffered changes are made permanent. There is no record that is associated with this position.

__FIRST

Positions to the first record in the access path that is currently being used by *fp*. The *key* parameter is ignored.

__KEY_EQ

Positions to the first record with the specified key.

__KEY_GE

Positions to the first record that has a key greater than or equal to the specified key.

__KEY_GT

Positions to the first record that has a key greater than the specified key.

__KEY_LE

Positions to the first record that has a key less than or equal to the specified key.

KEY LT

Positions to the first record that has a key less than the specified key.

__KEY_NEXTEQ

Positions to the next record that has a key equal to the key value with a length of *klen_rrn*, at the current position. The *key* parameter is ignored.

KEY NEXTUNO

Positions to the next record with a unique key from the current position in the access path. The *key* parameter is ignored.

KEY PREVEO

Positions to the previous record with a key equal to the key value with a length of *klen_rrn*, at the current position. The *key* parameter is ignored.

__KEY_PREVUNQ

Positions to the previous record with a unique key from the current position in the access path. The *key* parameter is ignored.

__LAST

Positions to the last record in the access path that is currently being used by *fp*. The *key* parameter is ignored.

__NEXT

Positions to the next record in the access path that is currently being used by *fp*. The *key* parameter is ignored.

__PREVIOUS

Positions to the previous record in the access path that is currently being used by *fp*. The *key* parameter is ignored.

__RRN_EQ

Positions to the record that has the relative record number specified on the *klen_rrn* parameter.

START

Positions to just before the first record in the file. There is no record that is associated with this position.

__START_FRC

Positions to just before the first record in a file. There is no record that is associated with this position. All buffered changes are made permanent.

__DATA_ONLY

Positions to data records only. Deleted records will be ignored.

__KEY_NULL_MAP

The NULL key map is to be considered when locating to a record by key.

__NO_LOCK

The record that is positioned will not be locked.

NO POSITION

The position of the file is not changed, but the located record will be locked if the file is open for update.

PRIOR

Positions to just before the requested record.

If you specify the start and end options (__START, __START_FRC, __END or __END_FRC) with any other options, the other options are ignored.

If you are positioned to __START or __END and perform a _Rreads operation, errno is set to EIOERROR.

Return Value

The _Rlocate() function returns a pointer to the _RIOFB_T structure associated with fp. If the _Rlocate() operation is successful, the num_bytes field contains 1. If _START, __START_FRC, _END or __END_FRC are specified, the num_bytes field is set to EOF. If the _Rlocate() operation is unsuccessful, the num_bytes field contains zero. The key and rrn fields are updated, and the key field will contain the complete key even if a partial key is specified.

The value of errno may be set to:

Table 5.

Value	Meaning
EBADKEYLN	The key length that is specified is not valid.
ENOTREAD	The file is not open for read operations
EIOERROR	A non-recoverable I/O error occurred.
EIORECERR	A recoverable I/O error occurred.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rlocate()

```
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
int main(void)
     RFILE *in;
            new purchase[21] = "PEAR
                                           1002022244";
    char
    /* Open the file for processing in keyed sequence.
                                                                    */
    if ( (in = Ropen("MYLIB/T1677RD4", "rr+, arrseq=N")) == NULL )
        printf("Open failed\n");
        exit(1);
    };
    /* Update the first record in the keyed sequence.
                                                                    */
    Rlocate(in, NULL, 0, FIRST);
    Rupdate(in, new purchase, 20);
    /* Force the end of data.
                                                                    */
    Rfeod(in);
    _Rclose(in);
```

• "_Ropen() — Open a Record File for I/O Operations"

_Ropen() — Open a Record File for I/O Operations

Format

```
#include <recio.h>
    RFILE * Ropen(const char * filename, const char * mode, ...);
```

Language Level: ILE C Extension

Description

The _Ropen() function opens the record file specified by *filename* according to the *mode* parameter, which may be followed by optional parameters, if the *varparm* keyword parameter is specified in the *mode* parameter. The open mode and keyword parameters may be separated by a comma and one or more spaces. The _Ropen() function does not dynamically create database files for you. All of the files you refer to in the _Ropen() function must exist, or the open operation will fail.

Files that are opened by the _Ropen() function are closed implicitly when the activation group they are opened in, is ended. If a pointer to a file opened in one activation group is passed to another activation group and the opening activation group is ended, the file pointer will no longer be valid.

The _Ropen() function applies to all types of files. The *filename* variable is any valid iSeries system file name.

The *mode* parameter specifies the type of access that is requested for the file. It contains an open mode that is followed by optional keyword parameters. The mode parameter may be one of the following values:

Mode Description

- Open an existing file for reading records. rr
- Open an existing file for writing records. If the file contains data, the wr content is cleared unless the file is a logical file.
- Open an existing file for writing records to the end of the file (append). ar
- Open an existing file for reading, writing or updating records. rr+
- Open an existing file for reading, writing or updating records. If the file wr+ contains data, the content is cleared unless the file is a logical file.
- Open an existing file for reading and writing records. All data is written to ar+ the end of the file.

The *mode* may be followed by any of the following keyword parameters:

Keyword

Description

arrseq=value

Where value can be:

- Y Specifies that the file is processed in arrival sequence.
- N Specifies that the file is processed using the access path that is used when the file was created. This is the default.

blkrcd=value

Where value can be:

- Y Performs record blocking. The iSeries system determines the most efficient block size for you. This parameter is valid for database, DDM, diskette and tape files. It is only valid for files opened for input-only or output-only (modes rr, wr, or ar).
- N Does not perform record blocking. This is the default.

ccsid=value

Specifies the CCSID that is used for translation of the file. The default is 0 which indicates that the job CCSID is used.

commit=value

Where value can be:

- Y Specifies that the database file is opened under commitment control. Commitment control must have been set up prior to this.
- N Specifies that the database file is not opened under commitment control. This is the default.

dupkey=value

value can be:

- Y Duplicate key values will be flagged in the _RIOFB_T structure.
- N Duplicate key values will not be flagged. This is the default.

indicators=value

Indicators are valid for printer, display, and ICF files. value can be:

- Y The indicators that are associated with the file are returned in a separate indicator area instead of in the I/O buffers.
- N The indicators are returned in the I/O buffers. This is the default.

lrecl=value

The length, in bytes, for fixed length records, and the maximum length for variable length records. This parameter is valid for diskette, display, printer, tape, and save files.

nullcap=value

Where value can be:

- Y The program is capable of handling null fields in records. This is valid for database and DDM files.
- N The program cannot handle null fields in records. This is the default.

riofb=value

Where value can be:

- Y All fields in the _RIOFB_T structure are updated by any I/O operation that returns a pointer to the _RIOFB_T structure. However, the blk_filled_by field is not updated when using the Rreadk function. This is the default.
- N Only the num_bytes field in the _RIOFB_T structure is updated.

rtncode=value

Where value can be:

- Y Use this option to bypass exception generation and handling. This will improve performance in the end-of-file and record-not-found cases. If the end-of-file is encountered, num_bytes will be set to EOF, but no errno values will be generated. If no record is found, num_bytes will be set to zero, and errno will be set to EIORECERR. This parameter is only valid for database and DDM files. For DDM files, num_bytes is not updated for _Rfeod.
- N The normal exception generation and handling process will occur for the cases of end-of-file and record-not-found. This is the default.

secure=value

Where value can be:

- Y Secures the file from overrides.
- N Does not secure the file from overrides. This is the default.

splfname=(value)

For spooled output only. Where value can be:

*FILE The name of the printer file is used for the spooled output file name.

spool-file-name

Specify the name of the spooled output file. A maximum of 10 characters can be used.

usrdta=(value)

To specify, for spooled output only, user-specified data that identifies the file.

user-data

Specify up to 10 characters of user-specified text.

varparm=(list)

Where (list) is a list of optional keywords indicating which optional parameters will be passed to Ropen(). The order of the keywords within the list indicates the order that the optional parameters will appear after the *mode* parameter. The following is a valid optional keyword:

lvlchk The lvlchk keyword is used in conjunction with the lvlchk option on #pragma mapinc. When this keyword is used, a pointer to an object of type _LVLCHK_T (generated by #pragma mapinc) must be specified after the mode parameter on the _Ropen() function. For more details on this pointer, see the lylchk option of #pragma mapinc in the WebSphere Development Studio: ILE C/C++ Programmer's Guide.

vlr=value

Variable length record, where value is the minimum length of bytes of a record to be written to the file. The value can equal -1, or range from 0 to the maximum record length of the file. This parameter is valid for database and DDM files.

When VLR processing is required, Ropen() will set min_length field. If the default value is not used, the minimum value that is provided by the user will be directly copied into min_length field. If the default value is specified, Ropen() gets the minimum length from DB portion of the open data path.

Return Value

The Ropen() function returns a pointer to a structure of type _RFILE if the file is opened successfully. It returns NULL if opening the file is unsuccessful.

The value of errno may be set to:

Value Meaning

EBADMODE

The file mode that is specified is not valid.

EBADNAME

The file name that is specified is not valid.

ENOTOPEN

The file is not open.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Ropen()

- "_Rclose() —Close a File" on page 229
- "<recio.h>" on page 9

_Ropnfbk() — Obtain Open Feedback Information

Format

```
#include <recio.h>
#include <xxfdbk.h>
_XXOPFB_T *_Ropnfbk(_RFILE *fp);
```

Language Level: ILE C Extension

Description

The _Ropnfbk() function returns a pointer to a copy of the open feedback area for the file that is specified by *fp*.

The _Ropnfbk() function is valid for all types of files.

Return Value

The _Ropnfbk() function returns NULL if an error occurs. See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses Ropnfbk()

```
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
int main(void)
    RFILE
               *fp;
   XXOPFB T
               *opfb;
    /* Open the file for processing in arrival sequence.
    if (( fp = _Ropen ( "MYLIB/T1677RD1", "rr+, arrseq=Y" )) == NULL )
       printf ( "Open failed\n" );
       exit (1);
    /* Get the library and file names of the file opened.
                                                                    */
    opfb = _Ropnfbk ( fp );
   printf ( "Library: %10.10s\nFile:
                                       %10.10s\n",
             opfb->library_name,
             opfb->file name);
    Rclose (fp);
```

• "_Rupfb() — Provide Information on Last I/O Operation" on page 290

_Rpgmdev() — Set Default Program Device

Format

```
#include <recio.h>
int Rpgmdev( RFILE *fp, char *dev);
```

Language Level: ILE C Extension

Thread Safe: NO.

Description

The Rpgmdev() function sets the current program device for the file that is associated with fp to dev. You must specify the device in uppercase.

The *dev* parameter is a null-ended C string.

The Rpgmdev() function is valid for display, ICF, and printer files.

Return Value

The Rpgmdev() function returns 1 if the operation is successful or zero if the device specified has not been acquired for the file. See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rpgmdev()

```
#include <stdio.h>
#include <recio.h>
#include <string.h>
#include <stdlib.h>
typedef struct {
    char name[20];
    char address[25];
} format1;
typedef struct {
    char name[8];
    char password[10];
} format2;
typedef union {
    format1 fmt1;
    format2 fmt2;
} formats;
int main(void)
    RFILE *fp; /* File pointer
    RIOFB T *rfb; /*Pointer to the file's feedback structure
    formats buf, in buf, out buf; /* Buffers to hold data
    /* Open the device file.
                                                                      */
    if (( fp = _{Ropen} ( "MYLIB/T1677RD2", "ar+" )) == NULL )
        printf ( "Could not open file\n" );
       exit ( 1 );
    Rpgmdev (fp, "DEVICE2");/* Change the default program device.
                              /* Replace with actual device name.
    _Rformat ( fp, "FORMAT2" ); /* Set the record format for the
                                 /* display file.
    rfb = _Rwrite ( fp, "", 0 ); /* Set up the display.
                                                                      */
   rfb = Rwriterd (fp, &buf, sizeof(buf));
    rfb = Rwrread (fp, &in buf, sizeof(in buf), &out buf,
                     sizeof(out buf ));
    /* Continue processing.
                                                                      */
    _Rclose ( fp );
```

- "_Racquire() —Acquire a Program Device" on page 228
- "_Rrelease() Release a Program Device" on page 283

_Rreadd() — Read a Record by Relative Record Number

Format

```
#include <recio.h>
_RIOFB_T *_Rreadd (_RFILE *fp, void *buf, size_t size,
                            int opts, long rrn);
```

Language Level: ILE C Extension

Thread Safe: YES. However, if the file pointer is passed among threads, the I/O feedback area is shared among those threads.

Description

1

The Rreadd() function reads the record that is specified by rrn in the arrival sequence access path for the file that is associated with fp. The _Rreadd() function locks the record specified by the rrn unless __NO_LOCK is specified. If the file is a keyed file, the keyed access path is ignored. Up to size number of bytes are copied from the record into *buf* (move mode only).

The following parameters are valid for the _Rreadd() function.

- buf Points to the buffer where the data that is read is to be stored. If locate mode is used, this parameter must be set to NULL.
- Specifies the number of bytes that are to be read and stored in buf. If locate size mode is used, this parameter is ignored.
- The relative record number of the record to be read. rrn
- Specifies the processing and access options for the file. The possible opts options are:

DFT

If the file is opened for updating, then the record being read is locked for update. The previously locked record will no longer be locked.

NO LOCK

Does not lock the record being positioned to.

The Rreadd() function is valid for database, DDM and display (subfiles) files.

Return Value

The Rreadd() function returns a pointer to the _RIOFB_T structure associated with fp. If the _Rreadd() operation is successful the num_bytes field is set to the number of bytes transferred from the system buffer to the user's buffer (move mode) or the record length of the file (locate mode). If blkrcd=Y and riofb=Y are specified, the blk_count and the blk_filled_by fields of the _RIOFB_T structure are updated. The key and rrn fields are also updated. If the file associated with fp is a display file, the sysparm field is updated. If it is unsuccessful, the num_bytes field is set to a value less than *size* and errno will be changed.

The value of errno may be set to:

Value Meaning

ENOTREAD

The file is not open for read operations.

ETRUNC

Truncation occurred on an I/O operation.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rreadd()

```
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
int main(void)
    RFILE
               *fp;
   XXOPFB T *opfb;
    /* Open the file for processing in arrival sequence.
    if (( fp = _Ropen ( "MYLIB/T1677RD1", "rr+, arrseq=Y" )) == NULL )
       printf ( "Open failed\n" );
       exit (1);
    /* Get the library and file names of the file opened.
                                                                     */
   opfb = _Ropnfbk ( fp );
   printf ( "Library: %10.10s\nFile:
                                       %10.10s\n",
             opfb->library name,
             opfb->file name);
   /* Get the second record.
                                                                     */
    _Rreadd ( fp, NULL, 20, __DFT, 2 );
   printf ( "Second record: $10.10s\n", *(fp->in buf) );
    Rclose (fp);
```

Related Information

- "_Rreadf() Read the First Record"
- "_Rreadindv() Read from an Invited Device" on page 267
- "_Rreadk() Read a Record by Key" on page 270
- "_Rreadl() Read the Last Record" on page 274
- "_Rreadn() Read the Next Record" on page 275
- "_Rreadnc() Read the Next Changed Record in a Subfile" on page 278
- "_Rreadp() Read the Previous Record" on page 279
- "_Rreads() Read the Same Record" on page 282

_Rreadf() — Read the First Record

Format

```
#include <recio.h>
RIOFB T * Rreadf ( RFILE *fp, void *buf, size t size, int opts);
```

Language Level: ILE C Extension

Thread Safe: YES. However, if the file pointer is passed among threads, the I/O feedback area is shared among those threads.

Description

The _Rreadf() function reads the first record in the access path that is currently being used for the file specified by *fp*. The access path may be keyed sequence or

arrival sequence. The Rreadf() function locks the first record unless __NO_LOCK is specified. Up to size number of bytes are copied from the record into buf (move mode only).

The following are valid parameters for the _Rreadf() function.

- buf This parameter points to the buffer where the data that is read is to be stored. If locate mode is used, this parameter must be set to NULL.
- size This parameter specifies the number of bytes that are to be read and stored in buf. If locate mode is used, this parameter is ignored.
- This parameter specifies the processing and access options for the file. The opts possible options are:

__DFT

If the file is opened for updating, then the record being read or positioned to is locked for update. The previously locked record will no longer be locked.

__NO_LOCK

Does not lock the record being positioned to.

The Rreadf() function is valid for database and DDM files.

Return Value

The Rreadf() function returns a pointer to the _RIOFB_T structure that is specified by fp. If the Rreadf() operation is successful the num_bytes field is set to the number of bytes transferred from the system buffer to the user's buffer (move mode) or the record length of the file (locate mode). The key and rrn fields are updated. If record blocking is taking place, the blk_count and blk_filled_by fields are updated. The *num_bytes* field is set to EOF if the file is empty. If it is unsuccessful, the *num_bytes* field is set to a value less than *size*, and errno is changed.

The value of errno may be set to:

Value Meaning

ENOTREAD

The file is not open for read operations.

ETRUNC

Truncation occurred on an I/O operation.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses Rreadf()

```
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
int main(void)
     RFILE
                *fp;
    XXOPFB T *opfb;
    /* Open the file for processing in arrival sequence.
    if (( fp = Ropen ( "MYLIB/T1677RD1", "rr+, arrseq=Y" )) == NULL )
        printf ( "Open failed\n" );
        exit ( 1 );
    /* Get the library and file names of the file opened.
                                                                        */
    opfb = _Ropnfbk ( fp );
printf ( "Library: %10.10s\nFile:
                                          %10.10s\n",
              opfb->library_name,
              opfb->file name);
    /* Get the first record.
                                                                        */
    _Rreadf ( fp, NULL, 20, __DFT );
    printf ("First record: %10.10s\n", *(fp->in buf));
    /* Delete the first record.
                                                                        */
    _Rdelete ( fp );
    _Rclose (fp);
```

- "_Rreadd() Read a Record by Relative Record Number" on page 263
- "_Rreadindv() Read from an Invited Device"
- "_Rreadk() Read a Record by Key" on page 270
- "_Rreadl() Read the Last Record" on page 274
- "_Rreadn() Read the Next Record" on page 275
- "_Rreadnc() Read the Next Changed Record in a Subfile" on page 278
- "_Rreadp() Read the Previous Record" on page 279
- "_Rreads() Read the Same Record" on page 282

_Rreadindv() — Read from an Invited Device

Format

```
#include <recio.h>
_RIOFB_T *_Rreadindv(_RFILE *fp, void *buf, size_t size, int opts);
```

Language Level: ILE C Extension

Thread Safe: NO.

Description

The Rreadindv() function reads data from an invited device.

The following are valid parameters for the _Rreadindv() function.

- buf Points to the buffer where the data that is read is to be stored. If locate mode is used, this parameter must be set to NULL.
- size Specifies the number of bytes that are to be read and stored in buf. If locate mode is used, this parameter is ignored.
- Specifies the processing options for the file. Possible values are: opts

__DFT

If the file is opened for updating, then the record being read or positioned to is locked. Otherwise, the option is ignored.

The Rreadindv() function is valid for display and ICF files.

Return Value

The _Rreadindv() function returns a pointer to the _RIOFB_T structure that is associated with fp. If the Rreadindv() function is successful, the num_bytes field is set to the number of bytes transferred from the system buffer to the user's buffer (move mode) or the record length of the file (locate mode). The sysparm and rrn (for subfiles) fields are also updated. The *num_bytes* field is set to EOF if the file is empty. If the Rreadindv() function is unsuccessful, the num_bytes field is set to a value less than the value of *size* and the errno will be changed.

The value of errno may be set to:

Value Meaning

ENOTREAD

The file is not open for read operations.

ETRUNC

Truncation occurred on an I/O operation.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses Rreadindv()

```
#include <stdio.h>
#include <recio.h>
#include <string.h>
#include <stdlib.h>
typedef struct {
    char name[20];
    char address[25];
} format1;
typedef struct {
   char name[8];
   char password[10];
} format2;
typedef union {
    format1 fmt1;
    format2 fmt2;
} formats;
int main(void)
{
    RFILE *fp;
                               /* File pointer
    RIOFB T *rfb;
                         /* Pointer to the file's feedback structure
    */
    XXIOFB T *iofb;
                              /* Pointer to the file's feedback area
    formats buf, in buf, out buf
                                          /* Buffers to hold data
 /* Open the device file.
                                                                   */
   if (( fp = _Ropen ( "MYLIB/T1677RD2", "ar+" )) == NULL )
       printf ( "Could not open file\n" );
       exit (1);
   Racquire ( fp,"DEVICE1" );
                                 /* Acquire another device. Replace */
                                  /* with actual device name.
   Rformat ( fp, "FORMAT1" );
                                  /* Set the record format for the
                                   /* display file.
   rfb = _{Rwrite} ( fp, "", 0 ); /* Set up the display.
    _Rpgmdev ( fp,"DEVICE2" ); /* Change the default program device. */
                               /* Replace with actual device name.
    Rformat (fp, "FORMAT2");
                               /* Set the record format for the
                                                                      */
                                /* display file.
                                                                      */
   rfb = Rwrite ( fp, "", 0 ); /* Set up the display.
                                                                      */
   rfb = _Rwriterd ( fp, &buf, sizeof(buf) );
   rfb = Rwrread (fp, &in buf, sizeof(in buf), &out buf,
                    sizeof(out buf ));
    _Rreadindv ( fp, &buf, sizeof(buf), __DFT );
                                  /* Read from the first device that */
                                  /* enters data - device becomes
                                                                     */
                                  /* default program device.
                                                                      */
/* Determine which terminal responded first.
    iofb = Riofbk ( fp );
    if ( !strncmp ( "FORMAT1 ", iofb -> rec_format, 10 ))
        Rrelease ( fp, "DEVICE1" );
   else
       _Rrelease(fp, "DEVICE2");
/* Continue processing.
                                                                   */
   printf ( "Data displayed is %45.45s\n", &buf);
    _Rclose (fp);
```

• "_Rreadd() — Read a Record by Relative Record Number" on page 263

- "_Rreadf() Read the First Record" on page 265
- "_Rreadk() Read a Record by Key"
- "_Rreadl() Read the Last Record" on page 274
- "_Rreadn() Read the Next Record" on page 275
- "_Rreadnc() Read the Next Changed Record in a Subfile" on page 278
- "_Rreadp() Read the Previous Record" on page 279
- "_Rreads() Read the Same Record" on page 282

_Rreadk() — Read a Record by Key

Format

```
#include <recio.h>
_RIOFB_T *_Rreadk(_RFILE *fp, void *buf, size_t size,
                   int opts, void *key, unsigned int keylen);
```

Language Level: ILE C Extension

Thread Safe: YES. However, if the file pointer is passed among threads, the I/O feedback area is shared among those threads.

Description

The Rreadk() function reads the record in the keyed access path that is currently being used for the file that is associated with fp. Up to size number of bytes are copied from the record into buf (move mode only). The Rreadk() function locks the record positioned to unless __NO_LOCK is specified. You must be processing the file using a keyed sequence path.

The following parameters are valid for the _Rreadk() function.

buf Points to the buffer where the data that is read is to be stored. If locate mode is used, this parameter must be set to NULL.

size Specifies the number of bytes that are to be read and stored in buf. If locate mode is used, this parameter is ignored.

key Points to the key to be used for reading.

Specifies the total length of the key to be used.

opts Specifies the processing options for the file. Possible values are:

```
DFT
      Default to __KEY_EQ.
```

KEY EQ

Positions to and reads the first record that has the specified key.

KEY GE

Positions to and reads the first record that has a key greater than or equal to the specified key.

KEY GT

Positions and reads to the first record that has a key greater than the specified key.

KEY LE

Positions to and reads the first record that has a key less than or equal to the specified key.

KEY LT

Positions to and reads the first record that has a key less than the specified key.

__KEY_NEXTEQ

Positions to and reads the next record that has a key equal to the key value at the current position. The *key* parameter is ignored.

__KEY_NEXTUNQ

Positions to and reads the next record with a unique key from the current position in the access path. The *key* parameter is ignored.

__KEY_PREVEQ

Positions to and reads the last record that has a key equal to the key value at the current position. The *key* parameter is ignored.

__KEY_PREVUNQ

Positions to and reads the previous record with a unique key from the current position in the access path. The *key* parameter is ignored.

__NO_LOCK

Do not lock the record for updating.

The positioning options are mutually exclusive.

The following options may be combined with the positioning options using the bit-wise OR (|) operator.

KEY NULL MAP

The NULL key map is to be considered when reading a record by key.

__NO_LOCK

The record that is positioned will not be locked.

The Rreadk() function is valid for database and DDM files.

Return Value

The _Rreadk() function returns a pointer to the _RIOFB_T structure associated with fp. If the _Rreadk() operation is successful the num_bytes field is set to the number of bytes transferred from the system buffer to the user's buffer (move mode) or the record length of the file (locate mode). The key and rrn fields will be updated. The key field will always contain the complete key if a partial key is specified. When using record blocking with _Rreadk(), only one record is read into the block. Thus there are zero records remaining in the block and the blk_count field of the _RIOFB_T structure will be updated with 0. The blk_filled_by field is not applicable to _Rreadk() and is not updated. If the record specified by key cannot be found, the num_bytes field is set to zero. If you are reading a record by a partial key, then the entire key is returned in the feedback structure. If it is unsuccessful, the num_bytes field is set to a value less than size and errno will be changed.

The value of errno may be set to:

Value Meaning

EBADKEYLN

The key length specified is not valid.

ENOTREAD

The file is not open for read operations.

ETRUNC

Truncation occurred on an I/O operation.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rreadk()

```
#include <stdio.h>
#include <recio.h>
#include <stdlib.h>
int main(void)
     RFILE *fp;
    RIOFB T *fb;
              buf[4];
    char
    /* Create a physical file
                                                                           */
    system("CRTPF FILE(QTEMP/MY FILE)");
    /* Open the file for write
                                                                           */
    if ( (fp = Ropen("QTEMP/MY FILE", "wr")) == NULL )
        printf("open for write fails\n");
        exit(1);
    /* write some records into the file
                                                                           */
    _Rwrite(fp, "KEY9", 4);
_Rwrite(fp, "KEY8", 4);
_Rwrite(fp, "KEY7", 4);
_Rwrite(fp, "KEY6", 4);
     __Rwrite(fp, "KEY5", 4);
     Rwrite(fp, "KEY4", 4);
    _Rwrite(fp, "KEY3", 4);
    _Rwrite(fp, "KEY2", 4);
_Rwrite(fp, "KEY1", 4);
    /* Close the file
                                                                           */
     Rclose(fp);
    7* Open the file for read
                                                                           */
    if ( (fp = _Ropen("QTEMP/MY_FILE", "rr")) == NULL )
        printf("open for read fails\n");
        exit(2);
    /* Read the record with key KEY3
                                                                           */
    fb = _Rreadk(fp, buf, 4, __KEY_EQ, "KEY3", 4);
    printf("record %d with value %4.4s\n", fb->rrn, buf);
    /* Read the next record with key less than KEY3
    fb = _Rreadk(fp, buf, 4, __KEY_LT, "KEY3", 4);
    printf("record %d with value %4.4s\n", fb->rrn, buf);
    /* Read the next record with key greater than KEY3
    fb = Rreadk(fp, buf, 4, KEY GT, "KEY3", 4);
    printf("record %d with value %4.4s\n", fb->rrn, buf);
    /* Read the next record with different key
    fb = _Rreadk(fp, buf, 4, __KEY_NEXTUNQ, "", 4);
    printf("record %d with value %4.4s\n", fb->rrn, buf);
    /* Close the file
                                                                           */
    Rclose(fp);
```

- "_Rreadd() Read a Record by Relative Record Number" on page 263
- "_Rreadf() Read the First Record" on page 265
- "_Rreadindv() Read from an Invited Device" on page 267
- "_Rreadl() Read the Last Record" on page 274
- "_Rreadn() Read the Next Record" on page 275
- "_Rreadnc() Read the Next Changed Record in a Subfile" on page 278
- "_Rreadp() Read the Previous Record" on page 279
- "_Rreads() Read the Same Record" on page 282

_RreadI() — Read the Last Record

Format

```
#include <recio.h>
RIOFB T * Rreadl ( RFILE *fp, void *buf, size t size, int opts);
```

Language Level: ILE C Extension

Thread Safe: YES. However, if the file pointer is passed among threads, the I/O feedback area is shared among those threads.

Description

The Rreadl () function reads the last record in the access path currently being used for the file specified by fp. The access path may be keyed sequence or arrival sequence. Up to size number of bytes are copied from the record into buf (move mode only). The Rread1() function locks the last record unless __NO_LOCK is specified.

The following parameters are valid for the Rread1() function.

buf Points to the buffer where the data that is read is to be stored. If locate mode is used, this parameter must be set to NULL.

size Specifies the number of bytes that are to be read and stored in buf. If locate mode is used, this parameter is ignored.

opts Specifies the processing options for the file. Possible values are:

DFT

If the file is opened for updating, then the record being read or positioned to is locked. The previously locked record will no longer be locked.

NO LOCK

Do not lock the record being positioned to.

The Rreadl() function is valid for database and DDM files.

Return Value

The Rreadl() function returns a pointer to the _RIOFB_T structure that is associated with fp. If the Rreadl () operation is successful the num_bytes field is set to the number of bytes transferred from the system buffer to the user's buffer (move mode) or the record length of the file (locate mode). The key and rrn fields will be updated. If record blocking is taking place, the blk_count and blk_filled_by fields will be updated. If the file is empty, the *num bytes* field is set to EOF. If it is unsuccessful, the *num_bytes* field is set to a value less than *size* and errno will be changed.

The value of errno may be set to:

Value Meaning

ENOTREAD

The file is not open for read operations.

ETRUNC

Truncation occurred on an I/O operation.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rreadl()

```
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
int main(void)
               *fp;
    RFILE
   XXOPFB T *opfb;
   /* Open the file for processing in arrival sequence.
   if (( fp = _Ropen ( "MYLIB/T1677RD1", "rr+, arrseq=Y" )) == NULL )
       printf ( "Open failed\n" );
       exit (1);
   /* Get the library and file names of the file opened.
                                                                     */
   opfb = _Ropnfbk ( fp );
   printf ( "Library: %10.10s\nFile: %10.10s\n",
             opfb->library name,
             opfb->file name);
   /* Get the last record.
                                                                     */
    Rreadl (fp, NULL, 20,
                            DFT );
   printf ("Last record: \sqrt[8]{10.10}s\n", *(fp->in buf));
    Rclose (fp);
```

Related Information

- "_Rreadd() Read a Record by Relative Record Number" on page 263
- "_Rreadf() Read the First Record" on page 265
- "_Rreadindv() Read from an Invited Device" on page 267
- "_Rreadk() Read a Record by Key" on page 270
- "_Rreadn() Read the Next Record"
- "_Rreadnc() Read the Next Changed Record in a Subfile" on page 278
- "_Rreadp() Read the Previous Record" on page 279
- "_Rreads() Read the Same Record" on page 282

Rreadn() — Read the Next Record

Format

```
#include <recio.h>
RIOFB T * Rreadn ( RFILE *fp, void *buf, size t size, int opts);
```

Language Level: ILE C Extension

Thread Safe: YES. However, if the file pointer is passed among threads, the I/O feedback area is shared among those threads.

Description

The Rreadn() function reads the next record in the access path that is currently being used for the file that is associated with fp. The access path may be keyed sequence or arrival sequence. Up to size number of bytes are copied from the record into buf (move mode only). The _Rreadn() function locks the record positioned to unless __NO_LOCK is specified.

If the file associated with fp is opened for sequential member processing and the current record position is the last record of any member in the file except the last, _Rreadn() will read the first record in the next member of the file.

If an Rlocate() operation positioned to a record specifying the __PRIOR option, Rreadn() will read the record positioned to by the Rlocate() operation.

If the file is open for record blocking and a call to Rreadp() has filled the block, the Rreadn() function is not valid if there are records remaining in the block. You can check the blk_count in _RIOFB_T to see if there are any remaining records.

The following are valid parameters for the Rreadn() function.

- Points to the buffer where the data that is read is to be stored. If locate buf mode is used, this parameter must be set to NULL.
- Specifies the number of bytes that are to be read and stored in buf. If locate size mode is used, this parameter is ignored.
- Specifies the processing options for the file. Possible values are: opts

__DFT

If the file is opened for updating, then the record being read or positioned to is locked. The previously locked record will no longer be locked.

NO LOCK

Do not lock the record being positioned to.

The Rreadn() function is valid for all types of files except printer files.

Return Value

The _Rreadn() function returns a pointer to the _RIOFB_T structure that is associated with fp. If the Rreadn() operation is successful the num_bytes field is set to the number of bytes transferred from the system buffer to the user's buffer (move mode) or the record length of the file (locate mode). The key and rrn fields are updated. If the file that is associated with fp is a display file, the sysparm field is also updated. If record blocking is taking place, the blk_count and the blk_filled_by fields of the _RIOFB_T structure are updated. If attempts are made to read beyond the last record in the file, the num_bytes field is set to EOF. If it is unsuccessful, the num bytes field is set to a value less than size, and errno is changed. If you are using device files and specify zero as the size, check errno to determine if the function was successful.

The value of errno may be set to:

Value Meaning

ENOTREAD

The file is not open for read operations.

ETRUNC

Truncation occurred on an I/O operation.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rreadn()

```
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
int main(void)
     RFILE
                *fp;
    XXOPFB T *opfb;
    /* Open the file for processing in arrival sequence.
    if (( fp = _Ropen ( "MYLIB/T1677RD1", "rr+, arrseq=Y" )) == NULL )
        printf ( "Open failed\n" );
        exit (1);
    }
    /* Get the library and file names of the file opened.
                                                                        */
    opfb = _Ropnfbk ( fp );
    printf ( "Library: %10.10s\nFile: %10.10s\n",
              opfb->library_name,
              opfb->file name);
    /* Get the first record.
                                                                        */
    _Rreadf ( fp, NULL, 20, __DFT ); printf ( "First record: \$10.10s\n", *(fp->in_buf) );
    /* Delete the second record.
                                                                        */
    Rreadn (fp, NULL, 20, DFT);
    _Rdelete (fp);
    Rclose (fp);
```

Related Information

- "_Rreadd() Read a Record by Relative Record Number" on page 263
- "_Rreadf() Read the First Record" on page 265
- "_Rreadindv() Read from an Invited Device" on page 267
- "_Rreadk() Read a Record by Key" on page 270
- "_Rreadl() Read the Last Record" on page 274
- "_Rreadnc() Read the Next Changed Record in a Subfile" on page 278
- "_Rreadp() Read the Previous Record" on page 279
- "_Rreads() Read the Same Record" on page 282

Rreadnc() — Read the Next Changed Record in a Subfile

Format

```
#include <recio.h>
RIOFB T * Rreadnc( RFILE *fp, void *buf, size t size);
```

Language Level: ILE C Extension

Thread Safe: NO.

Description

The Rreadnc() function reads the next changed record from the current position in the subfile that is associated with fp. The minimum size of data that is read from the screen are copied from the system buffer to buf.

The following are valid parameters for the Rreadnc() function.

Points to the buffer where the data that is read is to be stored. If locate mode is used, this parameter must be set to NULL.

size Specifies the number of bytes that are to be read and stored in buf.

The Rreadnc() function is valid for subfiles.

Return Value

The _Rreadnc() function returns a pointer to the _RIOFB_T structure that is associated with fp. If the Rreadnc() operation is successful the num_bytes field is set to the number of bytes transferred from the system buffer to the user's buffer (move mode) or the record length of the file (locate mode). The rrn and sysparm fields are updated. If there are no changed records between the current position and the end of the file, the num_bytes field is set to EOF. If it is unsuccessful, the num_bytes field is set to a value less than *size*, and errno is changed.

The value of errno may be set to:

Value Meaning

ENOTREAD

The file is not open for read operations.

ETRUNC

Truncation occurred on an I/O operation.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses Rreadnc()

```
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
#define LEN
                  10
#define NUM RECS
                 20
#define SUBFILENAME "MYLIB/T1677RD6"
                 "MYLIB/T1677RDB"
#define PFILENAME
typedef struct {
   char name[LEN];
   char phone[LEN];
} pf t;
#define RECLEN sizeof(pf t)
void init_subfile(_RFILE *, _RFILE *);
int main(void)
    RFILE
                  *pf;
    RFILE
                 *subf;
   /**************
    * Open the subfile and the physical file.
    if ((pf = Ropen(PFILENAME, "rr")) == NULL) {
      printf("can't open file %s\n", PFILENAME);
      exit(1);
   if ((subf = Ropen(SUBFILENAME, "ar+")) == NULL) {
      printf("can't open file %s\n", SUBFILENAME);
       exit(2);
   /**************
    * Initialize the subfile with records
    * from the physical file.
   init subfile(pf, subf);
   /**************
    \star Write the subfile to the display by writing \star
    * a record to the subfile control format.
    ************************************
    _Rformat(subf, "SFLCTL");
_Rwrite(subf, "", 0);
_Rreadnc(subf, "", 0);
    * Close the physical file and the subfile.
    ************************************
    Rclose(pf);
   Rclose(subf);
```

- "_Rreadd() Read a Record by Relative Record Number" on page 263
- "_Rreadf() Read the First Record" on page 265
- "_Rreadindv() Read from an Invited Device" on page 267
- "_Rreadk() Read a Record by Key" on page 270
- "_Rreadl() Read the Last Record" on page 274
- "_Rreadn() Read the Next Record" on page 275
- "_Rreadp() Read the Previous Record"
- "_Rreads() Read the Same Record" on page 282

Rreadp() — Read the Previous Record

Format

```
#include <recio.h>
RIOFB T * Rreadp( RFILE *fp, void *buf, size t size, int opts);
```

Language Level: ILE C Extension

Thread Safe: YES. However, if the file pointer is passed among threads, the I/O feedback area is shared among those threads.

Description

The _Rreadp() function reads the previous record in the access path that is currently being used for the file that is associated with fp. The access path may be keyed sequence or arrival sequence. Up to size number of bytes are copied from the record into buf (move mode only). The _Rreadp() function locks the record positioned to unless __NO_LOCK is specified.

If the file associated with fp is opened for sequential member processing and the current record position is the first record of any member in the file except the first, _Rreadp() will read the last record in the previous member of the file.

If the file is open for record blocking and a call to Rreadn() has filled the block, the _Rreadp() function is not valid if there are records remaining in the block. You can check the blk_count in _RIOFB_T to see if there are any remaining records.

The following are valid parameters for the Rreadp() function.

- buf Points to the buffer where the data that is read is to be stored. If locate mode is used, this parameter must be set to NULL.
- size Specifies the number of bytes that are to be read and stored in buf. If locate mode is used, this parameter is ignored.
- opts Specifies the processing options for the file. Possible values are:

__DFT

If the file is opened for updating, then the record being read or positioned to is locked. The previously locked record will no longer be locked.

NO LOCK

Do not lock the record being positioned to.

The _Rreadp() function is valid for database and DDM files.

Return Value

The _Rreadp() function returns a pointer to the _RIOFB_T structure that is associated with fp. If the Rreadp() operation is successful the num_bytes field is set to the number of bytes transferred from the system buffer to the user's buffer (move mode) or the record length of the file (locate mode). The key and rrn fields are also updated. If record blocking is taking place, the blk_count and the blk_filled_by fields of the _RIOFB_T structure are updated. If attempts are made to read prior to the first record in the file, the num_bytes field is set to EOF. If it is unsuccessful, the num_bytes field is set to a value less than size, and errno is changed.

The value of errno may be set to:

Value Meaning

ENOTREAD

The file is not open for read operations.

ETRUNC

Truncation occurred on an I/O operation.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rreadp()

```
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
int main(void)
     RFILE
                *fp;
    XXOPFB T *opfb;
    /* Open the file for processing in arrival sequence.
    if (( fp = Ropen ( "MYLIB/T1677RD1", "rr+, arrseq=Y" )) == NULL )
        printf ( "Open failed\n" );
        exit ( 1 );
    }
   /* Get the library and file names of the file opened.
                                                                     */
   opfb = Ropnfbk ( fp );
   printf ( "Library: %10.10s\nFile: %10.10s\n",
              opfb->library_name,
             opfb->file name);
    /* Get the last record.
                                                                     */
    _Rreadl ( fp, NULL, 20, __DFT );
   printf ("Last record: \sqrt[8]{10.10}s\n", *(fp->in buf));
    /* Get the previous record.
                                                                     */
    _Rreadp ( fp, NULL, 20, __DFT );
    printf ("Next to last record: %10.10s\n", *(fp->in_buf));
    _Rclose (fp);
```

Related Information

- "_Rreadd() Read a Record by Relative Record Number" on page 263
- "_Rreadf() Read the First Record" on page 265
- "_Rreadindv() Read from an Invited Device" on page 267
- "_Rreadk() Read a Record by Key" on page 270
- "_Rreadl() Read the Last Record" on page 274
- "_Rreadn() Read the Next Record" on page 275
- "_Rreadnc() Read the Next Changed Record in a Subfile" on page 278
- "_Rreads() Read the Same Record" on page 282

_Rreads() — Read the Same Record

Format

```
#include <recio.h>
RIOFB T * Rreads( RFILE *fp, void *buf, size t size, int opts);
```

Language Level: ILE C Extension

Thread Safe: YES. However, if the file pointer is passed among threads, the I/O feedback area is shared among those threads.

Description

The Rreads() function reads the current record in the access path that is currently being used for the file that is associated with fp. The access path may be keyed sequence or arrival sequence. Up to size number of bytes are copied from the record into buf (move mode only). The Rreads () function locks the record positioned to unless __NO_LOCK is specified.

If the current position in the file that is associated with fp has no record associated with it, the Rreads() function will fail.

The Rreads() function is not valid when the file is open for record blocking.

The following are valid parameters for the _Rreads() function.

buf Points to the buffer where the data that is read is to be stored. If locate mode is used, this parameter must be set to NULL.

Specifies the number of bytes that are to be read and stored in buf. If locate size mode is used, this parameter is ignored.

opts Specifies the processing options for the file. Possible values are:

DFT

If the file is opened for updating, then the record being read or positioned to is locked. The previously locked record will no longer be locked.

NO LOCK

Do not lock the record being positioned to.

The Rreads() function is valid for database and DDM files.

Return Value

The Rreads() function returns a pointer to the RIOFB T structure that is associated with fp. If the Rreads() operation is successful the num_bytes field is set to the number of bytes transferred from the system buffer to the user's buffer (move mode) or the record length of the file (locate mode). The key and rrn fields are also updated. If it is unsuccessful, the num_bytes field is set to a value less than size, and errno is changed.

The value of errno may be set to:

Value Meaning

ENOTREAD

The file is not open for read operations.

ETRUNC

Truncation occurred on an I/O operation.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rreads()

```
#include <stdlib.h>
#include <recio.h>
int main(void)
    RFILE
                *fp;
   XXOPFB T *opfb;
    /* Open the file for processing in arrival sequence.
   if (( fp = _Ropen ( "MYLIB/T1677RD1", "rr+, arrseq=Y" )) == NULL )
        printf ( "Open failed\n" );
        exit (1);
   /* Get the library and file names of the file opened.
                                                                      */
   opfb = Ropnfbk (fp);
   printf ( "Library: %10.10s\nFile: %10.10s\n",
             opfb->library name,
             opfb->file name);
    /* Get the last record.
    _Rreadl ( fp, NULL, 20, _
                             DFT );
   printf ("Last record: \sqrt[8]{10.10}s\n", *(fp->in buf));
   /* Get the same record without locking it.
    _Rreads ( fp, NULL, 20, __NO_LOCK);
   printf ( "Same record: \sqrt[8]{10.10}s\n", *(fp->in buf) );
    Rclose (fp);
```

Related Information

- "_Rreadd() Read a Record by Relative Record Number" on page 263
- "_Rreadf() Read the First Record" on page 265
- "_Rreadindv() Read from an Invited Device" on page 267
- "_Rreadk() Read a Record by Key" on page 270
- "_Rreadl() Read the Last Record" on page 274
- "_Rreadn() Read the Next Record" on page 275
- "_Rreadnc() Read the Next Changed Record in a Subfile" on page 278
- "_Rreadp() Read the Previous Record" on page 279

Rrelease() — Release a Program Device

Format

```
#include <recio.h>
int Rrelease( RFILE *fp, char *dev);
```

Language Level: ILE C Extension

Thread Safe: NO.

Description

The Rrelease() function releases the program device that is specified by dev from the file that is associated with fp. The device name must be specified in uppercase.

The *dev* parameter is a null-ended C string.

The Rrelease() function is valid for display and ICF files.

Return Value

The _Rrelease() function returns 1 if it is successful or zero if it is unsuccessful. The value of errno may be set to EIOERROR (a non-recoverable I/O error occurred) or EIORECERR (a recoverable I/O error occurred). See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rrelease()

```
#include <stdio.h>
#include <recio.h>
#include <string.h>
#include <stdlib.h>
typedef struct {
    char name[20];
   char address[25];
} format1;
typedef struct {
    char name[8];
    char password[10];
} format2;
typedef union {
    format1 fmt1;
    format2 fmt2;
} formats;
int main(void)
    _RFILE   *fp; /* File pointer
    _RIOFB_T *rfb; /*Pointer to the file's feedback structure
    XXIOFB T *iofb; /* Pointer to the file's feedback area
    formats buf, in buf, out buf; /* Buffers to hold data
 /* Open the device file.
    if (( fp = _Ropen ( "MYLIB/T1677RD2", "ar+" )) == NULL )
       printf ( "Could not open file\n" );
       exit (1);
    Racquire ( fp,"DEVICE1" );
                                  /* Acquire another device. Replace */
                                   /* with actual device name.
    Rformat ( fp, "FORMAT1" );
                                   /* Set the record format for the
                                   /* display file.
                                                                      */
    rfb = Rwrite ( fp, "", 0 );
                                 /* Set up the display.
                                                                      */
    _Rpgmdev ( fp, "DEVICE2" ); /* Change the default program device. */
                               /* Replace with actual device name.
    Rformat (fp, "FORMAT2");
                               /* Set the record format for the
```

```
/* display file.
   rfb = _Rwrite ( fp, "", 0 ); /* Set up the display.
rfb = _Rwriterd ( fp, &buf, sizeof(buf) );
rfb = _Rwrread ( fp, &in_buf, sizeof(in_buf), &out_buf,
                       sizeof(out buf ));
   _Rreadindv ( fp, &buf, sizeof(buf),
                                               DFT );
                                      /* Read from the first device that */
                                      /* enters data - device becomes
                                      /* default program device.
                                                                                 */
/* Determine which terminal responded first.
   iofb = Riofbk ( fp );
   if (!strncmp ("FORMAT1", iofb -> rec format, 10))
        _Rrelease ( fp, "DEVICE1" );
   else
        Rrelease(fp, "DEVICE2");
/* Continue processing.
                                                                             */
   printf ( "Data displayed is %45.45s\n", &buf);
   Rclose (fp);
```

• "_Racquire() —Acquire a Program Device" on page 228

_Rrisick() — Release a Record Lock

Format

```
#include <recio.h>
int Rrlslck( RFILE *fp);
```

Language Level: ILE C Extension

Description

The _Rrlslck() function releases the lock on the currently locked record for the file specified by *fp*. The file must be open for update, and a record must be locked. If the _NO_POSITION option was specified on the _Rlocate() operation that locked the record, the record released may not be the record currently positioned to.

The Rrlslck() function is valid for database and DDM files.

Return Value

The _Rrlslck() function returns 1 if the operation is successful, or zero if the operation is unsuccessful.

The value of errno may be set to:

Value Meaning

ENOTUPD

The file is not open for update operations.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rrlslck()

```
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
int main(void)
    char
                buf[21];
    _RFILE
                *fp;
    _XXOPFB_T
               *opfb;
                result;
   /* Open the file for processing in arrival sequence.
   if (( fp = _Ropen ( "MYLIB/T1677RD1", "rr+, arrseq=Y" )) == NULL )
        printf ( "Open failed\n" );
        exit (1);
    };
    /* Get the library and file names of the file opened.
                                                                      */
    opfb = Ropnfbk (fp);
    printf ( "Library: %10.10s\nFile:
                                         %10.10s\n",
              opfb->library_name,
              opfb->file_name);
    /* Get the last record.
                                                                      */
    _Rreadl ( fp, NULL, 20, __DFT );
   printf ( "Last record: \sqrt[8]{10}.10s\n", *(fp->in_buf) );
    /* Rrlslck example.
                                                                      */
   result = _Rrlslck ( fp );
    if (result == 0)
       printf(" Rrlslck failed.\n");
    _Rclose (fp);
```

Related Information

• "_Rdelete() —Delete a Record" on page 232

_Rrollbck() — Roll Back Commitment Control Changes

Format

```
#include <recio.h>
int Rrollbck(void);
```

Language Level: ILE C Extension

Thread Safe: NO.

Description

The _Rrollbck() function reestablishes the last commitment boundary as the current commitment boundary. All changes that are made to the files under commitment control in the job, are reversed. All locked records are released. Any file that is open under commitment control in the job will be affected. You must

specify the keyword parameter commit=y when the file is opened to be under commitment control. A commitment control environment must have been set up prior to this.

The _Rrollbck() function is valid for database and DDM files.

Return Value

The _Rrollbck() function returns 1 if the operation is successful or zero if the operation is unsuccessful. The value of errno may be set to EIOERROR (a non-recoverable I/O error occurred) or EIORECERR (a recoverable I/O error occurred). See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rrollbck() #include <stdio.h> #include <recio.h> #include <stdlib.h> #include <string.h> int main(void) char buf[40]; rc = 1;int RFILE *purf; RFILE *dailyf; /* Open purchase display file and daily transaction file if ((purf = Ropen ("MYLIB/T1677RD3", "ar+,indicators=y")) == NULL) printf ("Display file did not open.\n"); exit (1); if ((dailyf = Ropen ("MYLIB/T1677RDA", "wr,commit=y")) == NULL) printf ("Daily transaction file did not open.\n"); exit (2); /* Select purchase record format */ Rformat (purf, "PURCHASE"); /* Invite user to enter a purchase transaction. /* The Rwrite function writes the purchase display. */ _Rwrite (purf, "", 0); _Rreadn (purf, buf, sizeof(buf), __DFT); /* Update daily transaction file rc = ((_Rwrite (dailyf, buf, sizeof(buf)))->num_bytes); /* If the databases were updated, then commit the transaction. /* Otherwise, rollback the transaction and indicate to the /* user that an error has occurred and end the application. if (rc) { _Rcommit ("Transaction complete"); } else { Rrollbck ();

_Rformat (purf, "ERROR");

}

```
Rclose ( purf );
Rclose (dailyf);
```

- "_Rcommit() —Commit Current Record" on page 230
- Backup and Recovery manual

_Rupdate() — Update a Record

Format

```
#include <recio.h>
RIOFB T * Rupdate( RFILE *fp, void *buf, size t size);
```

Language Level: ILE C Extension

Thread Safe: YES. However, if the file pointer is passed among threads, the I/O feedback area is shared among those threads.

Description

The Rupdate() function updates the record that is currently locked for update in the file that is specified by fp. The file must be open for update. A record is locked for update by reading or locating to it unless __NO_LOCK is specified on the read or locate operation. If the __NO_POSITION option is specified on a locate operation the record updated may not be the record currently positioned to. After the update operation, the updated record is no longer locked.

The number of bytes that are copied from buf to the record is the minimum of size and the record length of the file (move mode only). If size is greater than the record length, the data is truncated, and errno is set to ETRUNC. One complete record is always written to the file. If the size is less than the record length of the file, the remaining data in the record will be the original data that was read into the system buffer by the read that locked the record. If a locate operation locked the record, the remaining data will be what was in the system input buffer prior to the locate.

The Rupdate() function can be used to update deleted records and key fields. A deleted record that is updated will no longer be marked as a deleted record. In both of these cases any keyed access paths defined for fp will be changed.

Note: If locate mode is being used, _Rupdate() works on the data in the file's input buffer.

The Rupdate() function is valid for database, display (subfiles) and DDM files.

Return Value

The Rupdate() function returns a pointer to the _RIOFB_T structure associated with fp. If the Rupdate() function is successful, the num_bytes field is set to the number of bytes transferred from the system buffer to the user's buffer (move mode) or the record length of the file (locate mode). If fp is a display file, the sysparm field is updated. If the Rupdate() function is unsuccessful, the

num_bytes field is set to a value less than the *size* specified (move mode) or zero (locate mode). The errno value will also be changed.

The value of errno may be set to:

Value Meaning

ENOTUPD

The file is not open for update operations.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rupdate()

```
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
int main(void)
    RFILE *in;
   char new purchase[21] = "PEAR 1002022244";
   /* Open the file for processing in keyed sequence.
                                                                   */
   if ( (in = Ropen("MYLIB/T1677RD4", "rr+, arrseq=N")) == NULL )
       printf("Open failed\n");
       exit(1);
   };
    /* Update the first record in the keyed sequence.
                                                                   */
    _Rlocate(in, NULL, 0, __FIRST);
   _Rupdate(in, new_purchase, 20);
   /* Force the end of data.
    Rfeod(in);
    Rclose(in);
}
```

Related Information

- "_Rreadd() Read a Record by Relative Record Number" on page 263
- "_Rreadf() Read the First Record" on page 265
- "_Rreadindv() Read from an Invited Device" on page 267
- "_Rreadk() Read a Record by Key" on page 270
- "_Rreadl() Read the Last Record" on page 274
- "_Rreadn() Read the Next Record" on page 275
- "_Rreadnc() Read the Next Changed Record in a Subfile" on page 278
- "_Rreadp() Read the Previous Record" on page 279
- "_Rreads() Read the Same Record" on page 282

_Rupfb() — Provide Information on Last I/O Operation

Format

```
#include <recio.h>
RIOFB T * Rupfb( RFILE *fp);
```

Language Level: ILE C Extension

Thread Safe: YES. However, if the file pointer is passed among threads, the I/O feedback area is shared among those threads.

Description

The Rupfb() function updates the feedback structure associated with the file specified by fp with information about the last I/O operation. The _RIOFB_T structure will be updated even if riofb=N was specified when the file was opened. The num bytes field of the RIOFB T structure will not be updated. See "<recio.h>" on page 9 for a description of the _RIOFB_T structure.

The Rupfb() function is valid for all types of files.

Return Value

The Rupfb() function returns a pointer to the _RIOFB_T structure specified by fp. See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses Rupfb()

```
#include <stdio.h>
#include <recio.h>
#include <stdlib.h>
int main(void)
     RFILE *fp;
     RIOFB T *fb;
    /* Create a physical file
                                                                                 */
    system("CRTPF FILE(QTEMP/MY FILE) RCDLEN(80)");
    /* Open the file for write
                                                                                 */
    if ( (fp = Ropen("QTEMP/MY FILE", "wr")) == NULL )
         printf("open for write fails\n");
         exit(1);
    /* Write some records into the file
                                                                                 */
    _Rwrite(fp, "This is record 1", 16);
_Rwrite(fp, "This is record 2", 16);
_Rwrite(fp, "This is record 3", 16);
     Rwrite(fp, "This is record 4", 16);
     _Rwrite(fp, "This is record 5", 16);
     _Rwrite(fp, "This is record 6", 16);
     _Rwrite(fp, "This is record 7", 16);
     _Rwrite(fp, "This is record 8", 16);
_Rwrite(fp, "This is record 9", 16);
    /* Close the file
                                                                                 */
     Rclose(fp);
    /* Open the file for read
    if ( (fp = Ropen("QTEMP/MY FILE", "rr, blkrcd = y")) == NULL )
         printf("open for read fails\n");
         exit(2);
```

```
/* Read some records
                                                            */
_Rreadn(fp, NULL, 80, __DFT);
_Rreadn(fp, NULL, 80, __DFT);
/* Call Rupfb and print feed back information
                                                            */
fb = Rupfb(fp);
printf("record number ----- %d\n",
      fb->rrn);
printf("number of bytes read ----- %d\n",
      fb->num bytes);
printf("number of records remaining in block --- %hd\n",
      fb->blk count);
if ( fb->blk_filled_by == __READ_NEXT )
   printf("block filled by ----- READ NEXT\n");
else
   printf("block filled by ----- READ PREV\n");
                                                            */
/* Close the file
Rclose(fp);
```

• "_Ropnfbk() — Obtain Open Feedback Information" on page 261

_Rwrite() — Write the Next Record

Format

I

```
#include <recio.h>
RIOFB T * Rwrite( RFILE *fp, void *buf, size t size);
```

Language Level: ILE C Extension

Thread Safe: YES. However, if the file pointer is passed among threads, the I/O feedback area is shared among those threads.

Description

The _Rwrite() function has two modes: move and locate. When *buf* points to a user buffer, _Rwrite() is in move mode. When *buf* is NULL, the function is in locate mode.

The _Rwrite() function appends a record to the file specified by *fp*. The number of bytes copied from *buf* to the record is the minimum of *size* and the record length of the file (move mode only). If size is greater than the record length, the data is truncated and errno is set to ETRUNC. One complete record is always written if the operation is successful.

If you are using _Ropen() and then _Rwrite() to output records to a source physical file, the sequence numbers must be manually appended.

The _Rwrite() function has no effect on the position of the file for a subsequent read operation.

If record blocking is taking place and the file associated with *fp* is nearing the limit of the number of records it can contain, records may be lost although the _Rwrite() function indicates success. This can happen if another file pointer is

being used to write records to the file and it fills the file before the records in the block are written to the file. In this case, the Rwrite() function will indicate an error has occurred only on the call to the _Rwrite() function that sends the data to the database.

The Rwrite() function is valid for all types of files.

Return Value

The _Rwrite() function returns a pointer to the _RIOFB_T structure that is associated with fp. If the Rwrite() operation is successful the num_bytes field is set to the number of bytes written for both move mode and locate mode. The function transfers the bytes from the user's buffer to the system buffer. If record blocking is taking place, the function only updates the rrn and key fields when it sends the block to the database. If fp is a display, ICF or printer file, the function updates the sysparm field. If it is unsuccessful, the num_bytes field is set to a value less than size specified (move mode) or zero (locate mode) and errno is changed.

The value of errno may be set to:

Value Meaning

ENOTWRITE

The file is not open for write operations.

ETRUNC

Truncation occurred on an I/O operation.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses Rwrite()

```
#include <stdio.h>
#include <recio.h>
#include <string.h>
#include <stdlib.h>
typedef struct {
   char name[20];
    char address[25];
} format1;
typedef struct {
    char name[8];
   char password[10];
} format2;
typedef union {
    format1 fmt1;
    format2 fmt2:
} formats;
int main(void)
    RFILE
           *fp; /* File pointer
    RIOFB T *rfb; /*Pointer to the file's feedback structure
    XXIOFB T *iofb; /* Pointer to the file's feedback area
    formats buf, in buf, out buf; /* Buffers to hold data
 /* Open the device file.
```

```
if (( fp = _Ropen ( "MYLIB/T1677RD2", "ar+" )) == NULL )
      printf ( "Could not open file\n" );
      exit (1);
   Racquire ( fp,"DEVICE1" );
                               /* Acquire another device. Replace */
                               /* with actual device name.
  Rformat (fp, "FORMAT1");
                               /* Set the record format for the
                               /* display file.
                                                               */
  /* Replace with actual device name.
   Rformat ( fp, "FORMAT2" );
                            /* Set the record format for the
                                                               */
                             /* display file.
                                                                */
  rfb = Rwrite (fp, "", 0); /* Set up the display.
                                                                */
  rfb = _Rwriterd ( fp, &buf, sizeof(buf) );
  rfb = _Rwrread ( fp, &in_buf, sizeof(in_buf), &out_buf,
                  sizeof(out buf ));
  _Rreadindv ( fp, &buf, sizeof(buf), __DFT );
                              /* Read from the first device that */
                              /* enters data - device becomes
                              /* default program device.
/* Determine which terminal responded first.
  iofb = Riofbk (fp);
   if (!strncmp ("FORMAT1", iofb -> rec format, 10))
      _Rrelease ( fp, "DEVICE1" );
  else
      _Rrelease(fp, "DEVICE2");
/* Continue processing.
                                                             */
  printf ( "Data displayed is %45.45s\n", &buf);
   Rclose (fp);
```

- "_Rwrited() Write a Record Directly"
- "_Rwriterd() Write and Read a Record" on page 296
- "Rwrread() Write and Read a Record (separate buffers)" on page 297

_Rwrited() — Write a Record Directly

Format

```
#include <recio.h>
RIOFB T * Rwrited( RFILE *fp, void *buf, size t size, unsigned long rrn);
```

Language Level: ILE C Extension

Thread Safe: YES. However, if the file pointer is passed among threads, the I/O feedback area is shared among those threads.

Description

The _Rwrited() function writes a record to the file associated with *fp* at the position specified by *rrn*. The _Rwrited() function will only write over deleted records. The number of bytes copied from *buf* to the record is the minimum of *size* and the record length of the file (move mode only). If size is greater than the

record length, the data is truncated, and errno is set to ETRUNC. One complete record is always written if the operation is successful.

The Rwrited() function has no effect on the position of the file for a read operation.

The _Rwrited() function is valid for database, DDM and subfiles.

Return Value

The Rwrited() function returns a pointer to the _RIOFB_T structure associated with fp. If the _Rwrited() operation is successful the num_bytes field is set to the number of bytes transferred from the user's buffer to the system buffer (move mode) or the record length of the file (locate mode). The rrn field is updated. If fp is a display file, the sysparm field is updated. If it is unsuccessful, the num_bytes field is set to a value less than size specified (move mode) or zero (locate mode) and errno is changed.

The value of errno may be set to:

Value Meaning

ENOTWRITE

The file is not open for write operations.

ETRUNC

Truncation occurred on an I/O operation.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rwrited()

```
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
#define LEN
                      10
#define NUM RECS
                      20
#define SUBFILENAME "MYLIB/T1677RD6"
                      "MYLIB/T1677RDB"
#define PFILENAME
typedef struct {
    char name[LEN];
    char phone[LEN];
} pf t;
#define RECLEN sizeof(pf t)
void init_subfile(_RFILE *, _RFILE *);
int main(\overline{void})
     RFILE
                       *pf;
     RFILE
                      *subf;
/* Open the subfile and the physical file.
    if ((pf = Ropen(PFILENAME, "rr")) == NULL) {
    printf("can't open file %s\n", PFILENAME);
        exit(1);
    if ((subf = _Ropen(SUBFILENAME, "ar+")) == NULL) {
        printf("can't open file %s\n", SUBFILENAME);
        exit(2);
/* Initialize the subfile with records
  * from the physical file.
                                                       */
    init subfile(pf, subf);
/* Write the subfile to the display by writing
  * a record to the subfile control format.
     _Rformat(subf, "SFLCTL");
_Rwrite(subf, "", 0);
_Rreadnc(subf, "", 0);
/* Close the physical file and the subfile.
                                                      */
    _Rclose(pf);
    _Rclose(subf);
void init subfile( RFILE *pf, RFILE *subf)
         RIOFB_T
                        *fb;
        int
                        i;
        pf t
                        record;
/* Select the subfile record format.
                                                      */
         _Rformat(subf, "SFL");
        for (i = 1; i <= NUM_RECS; i++)
             fb = Rreadn(pf, &record, RECLEN, __DFT);
             if (fb->num bytes != RECLEN)
                                             {
             printf("%d\n", fb->num_bytes);
             printf("%d\n", RECLEN);
                 printf("error occurred during read\n");
                 exit(3);
             fb = Rwrited(subf, &record, RECLEN, i);
             if (fb->num bytes != RECLEN) {
                 printf("error occurred during write\n");
                 exit(4);
        }
    }
```

- "_Rwrite() Write the Next Record" on page 291
- "_Rwriterd() Write and Read a Record" on page 296
- "_Rwrread() Write and Read a Record (separate buffers)" on page 297

_Rwriterd() — Write and Read a Record

Format

```
#include <recio.h>
_RIOFB_T *_Rwriterd(_RFILE *fp, void *buf, size_t size);
```

Language Level: ILE C Extension

Thread Safe: NO.

Description

The _Rwriterd() function performs a write and then a read operation on the file that is specified by fp. The minimum of size and the length of the current record format determines the amount of data to be copied between the system buffer and buf for both the write and read parts of the operation. If size is greater than the record length of the current format, errno is set to ETRUNC on the write part of the operation. If size is less than the length of the current record format, errno is set to ETRUNC on the read part of the operation.

The _Rwriterd() function is valid for display and ICF files.

Return Value

The Rwriterd() function returns a pointer to the _RIOFB_T structure that is associated with fp. If the Rwriterd() operation is successful, the num_bytes field is set to the number of bytes transferred from the system buffer to buf on the read part of the operation (move mode) or the record length of the file (locate mode).

The value of errno may be set to:

Value Meaning

ENOTUPD

The file is not open for update operations.

ETRUNC

Truncation occurred on an I/O operation.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rwriterd()

```
#include <stdio.h>
#include <recio.h>
#include <string.h>
#include <stdlib.h>
typedef struct {
    char name[20];
    char address[25];
} format1;
typedef struct {
    char name[8];
    char password[10];
} format2;
typedef union {
    format1 fmt1;
    format2 fmt2;
} formats;
int main(void)
    RFILE *fp; /* File pointer
    RIOFB T *rfb; /*Pointer to the file's feedback structure
    formats buf, in buf, out buf; /* Buffers to hold data
    /* Open the device file.
                                                                      */
    if (( fp = _{Ropen} ( "MYLIB/T1677RD2", "ar+" )) == NULL )
        printf ( "Could not open file\n" );
        exit ( 1 );
    Rpgmdev (fp, "DEVICE2");/* Change the default program device.
                              /* Replace with actual device name.
    _Rformat ( fp,"FORMAT2" ); /* Set the record format for the
                                 /* display file.
    rfb = _Rwrite ( fp, "", 0 ); /* Set up the display.
                                                                      */
   rfb = Rwriterd (fp, &buf, sizeof(buf));
    rfb = Rwrread (fp, &in buf, sizeof(in buf), &out buf,
                     sizeof(out buf ));
    /* Continue processing.
                                                                      */
    _Rclose ( fp );
```

- "_Rwrite() Write the Next Record" on page 291
- "_Rwrited() Write a Record Directly" on page 293
- "_Rwrread() Write and Read a Record (separate buffers)"

_Rwrread() — Write and Read a Record (separate buffers)

Format

Language Level: ILE C Extension

Thread Safe: NO.

Description

The _Rwrread() function performs a write and then a read operation on the file that is specified by fp. Separate buffers may be specified for the input and output data. The minimum of size and the length of the current record format determines the amount of data to be copied between the system buffer and the buffers for both the write and read parts of the operation. If out_buf_size is greater than the record length of the current format, errno is set to ETRUNC on the write part of the operation. If *in_buf_size* is less than the length of the current record format, errno is set to ETRUNC on the read part of the operation.

The Rwrread() function is valid for display and ICF files.

Return Value

The _Rwrread() function returns a pointer to the _RIOFB_T structure that is associated with fp. If the Rwrread() operation is successful, the num_bytes field is set to the number of bytes transferred from the system buffer to *in_buf* in the read part of the operation (move mode) or the record length of the file (locate mode).

The value of errno may be set to:

Value Meaning

ENOTUPD

The file is not open for update operations.

ETRUNC

Truncation occurred on an I/O operation.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

See Table 12 on page 451 and Table 14 on page 455 for errno settings.

Example that uses _Rwrread()

```
#include <stdio.h>
#include <recio.h>
#include <string.h>
#include <stdlib.h>
typedef struct {
    char name[20];
    char address[25];
} format1;
typedef struct {
    char name[8];
    char password[10];
} format2;
typedef union {
    format1 fmt1;
    format2 fmt2;
} formats;
int main(void)
    RFILE *fp; /* File pointer
    RIOFB T *rfb; /*Pointer to the file's feedback structure
   formats buf, in buf, out buf; /* Buffers to hold data
    /* Open the device file.
                                                                      */
    if (( fp = _Ropen ( "MYLIB/T1677RD2", "ar+" )) == NULL )
        printf ( "Could not open file\n" );
       exit ( 1 );
    Rpgmdev (fp, "DEVICE2");/* Change the default program device.
                              /* Replace with actual device name.
    _Rformat ( fp, "FORMAT2" ); /* Set the record format for the
                                 /* display file.
   rfb = _Rwrite ( fp, "", 0 ); /* Set up the display.
                                                                      */
   rfb = Rwriterd (fp, &buf, sizeof(buf));
    rfb = Rwrread (fp, &in buf, sizeof(in buf), &out buf,
                    sizeof(out buf ));
    /* Continue processing.
                                                                      */
    Rclose (fp);
```

- "_Rwrite() Write the Next Record" on page 291
- "_Rwrited() Write a Record Directly" on page 293
- "_Rwriterd() Write and Read a Record" on page 296

scanf() — Read Data

Format

```
#include <stdio.h>
int scanf(const char *format-string, argument-list);
```

Language Level: ANSI

Description

The scanf() function reads data from the standard input stream stdin into the locations that is given by each entry in argument-list. Each argument must be a pointer to a variable with a type that corresponds to a type specifier in format-string. The format-string controls the interpretation of the input fields, and is a multibyte character string that begins and ends in its initial shift state.

The *format-string* can contain one or more of the following:

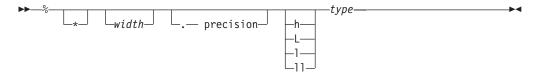
- White-space characters, as specified by the isspace() function (such as blanks and new-line characters). A white-space character causes the scanf() function to read, but not to store, all consecutive white-space characters in the input up to the next character that is not white space. One white-space character in format-string matches any combination of white-space characters in the input.
- Characters that are not white space, except for the percent sign character (%). A non-whitespace character causes the scanf() function to read, but not to store, a matching non-whitespace character. If the next character in stdin does not match, the scanf() function ends.
- Format specifications, introduced by the percent sign (%). A format specification causes the scanf() function to read and convert characters in the input into values of a specified type. The value is assigned to an argument in the argument

The scanf() function reads format-string from left to right. Characters outside of format specifications are expected to match the sequence of characters in stdin; the matched characters in stdin are scanned but not stored. If a character in stdin conflicts with format-string, scanf() ends. The conflicting character is left in stdin as if it had not been read.

When the first format specification is found, the value of the first input field is converted according to the format specification and stored in the location specified by the first entry in argument-list. The second format specification converts the second input field and stores it in the second entry in argument-list, and so on through the end of format-string.

An input field is defined as all characters up to the first white-space character (space, tab, or new line), up to the first character that cannot be converted according to the format specification, or until the field width is reached, whichever comes first. If there are too many arguments for the format specifications, the extra arguments are ignored. The results are undefined if there are not enough arguments for the format specifications.

A format specification has the following form:



Each field of the format specification is a single character or a number signifying a particular format option. The type character, which appears after the last optional format field, determines whether the input field is interpreted as a character, a string, or a number. The simplest format specification contains only the percent sign and a type character (for example, %s).

١

Each field of the format specification is discussed in detail below. If a percent sign (%) is followed by a character that has no meaning as a format control character, that character and following characters up to the next percent sign are treated as an ordinary sequence of characters; that is, a sequence of characters that must match the input. For example, to specify a percent-sign character, use %%.

The following restrictions apply to pointer printing and scanning:

- If a pointer is printed out and scanned back from the same activation group, the scanned back pointer will be compared equal to the pointer that is printed out.
- If a scanf() family function scans a pointer that was printed out by a different activation group, the scanf() family function will set the pointer to NULL.

See the WebSphere Development Studio: ILE C/C++ Programmer's Guide for more information about using iSeries pointers.

An asterisk (*) following the percent sign suppresses assignment of the next input field, which is interpreted as a field of the specified *type*. The field is scanned but not stored.

The *width* is a positive decimal integer controlling the maximum number of characters to be read from stdin. No more than *width* characters are converted and stored at the corresponding *argument*. Fewer than *width* characters are read if a white-space character (space, tab, or new line), or a character that cannot be converted according to the given format occurs before *width* is reached.

The optional size modifiers h, l, ll, and L indicate the size of the receiving object. The conversion characters d, i, and n must be preceded by h if the corresponding argument is a pointer to short int rather than a pointer to int, by l if it is a pointer to long int, or by ll if it is a pointer to long long int. Similarly, the conversion characters o, u, x, and X must be preceded by h if the corresponding argument is a pointer to unsigned short int rather than a pointer to unsigned int, by l if it is a pointer to unsigned long int, or by ll if it is a pointer to unsigned long long int. The conversion characters e, E, f, g, and G must be preceded by l if the corresponding argument is a pointer to double rather than a pointer to float, or by L if it is a pointer to a long double. Finally, the conversion characters c, s, and [must be preceded by l if the corresponding argument is a pointer to wchar_t rather than a pointer to a single byte character type. If an h, l, L or ll appears with any other conversion character, the behavior is undefined.

The *type* characters and their meanings are in the following table:

Character	Type of Input Expected	Type of Argument
d	Signed decimal integer	Pointer to int.
o	Unsigned octal integer	Pointer to unsigned int.
x, X	Unsigned hexadecimal integer	Pointer to unsigned int.
i	Decimal, hexadecimal, or octal integer	Pointer to int.
u	Unsigned decimal integer	Pointer to unsigned int.
e, f, g, E, G	Floating-point value consisting of an optional sign (+ or -); a series of one or more decimal digits possibly containing a decimal point; and an optional exponent (e or E) followed by a possibly signed integer value.	Pointer to float.

Character	Type of Input Expected	Type of Argument
D(n,p)	Packed decimal value consisting of an optional sign (+ or -); then a non-empty sequence of digits, optionally a series of one or more decimal digits possibly containing a decimal point, but not a decimal suffix. The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-whitespace character, in the expected form. It contains no characters if the input string is empty or consists entirely of white space, or if the first non-whitespace character is anything other than a sign, a digit, or a decimal point character.	Pointer to decimal(n,p). Since the internal representation of the binary coded decimal object is the same as the internal representation of the packed decimal data type, you can use the type character D(n,p).
С	Character; white-space characters that are ordinarily skipped are read when c is specified	Pointer to char large enough for input field.
S	String	Pointer to character array large enough for input field plus a ending null character (\0), which is automatically appended.
n	No input read from stream or buffer	Pointer to int, into which is stored the number of characters successfully read from the <i>stream</i> or buffer up to that point in the call to scanf().
р	Pointer to void converted to series of characters	Pointer to void.
lc	Multibyte character constant	Pointer to wchar_t.
ls	Multibyte string constant	Pointer to wchar_t string.

To read strings not delimited by space characters, substitute a set of characters in brackets ([]) for the s (string) type character. The corresponding input field is read up to the first character that does not appear in the bracketed character set. If the first character in the set is a caret (), the effect is reversed: the input field is read up to the first character that does appear in the rest of the character set.

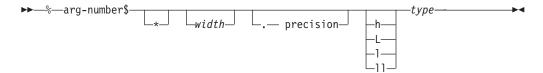
To store a string without storing an ending null character (\0), use the specification %ac, where a is a decimal integer. In this instance, the c type character means that the argument is a pointer to a character array. The next a characters are read from the input stream into the specified location, and no null character is added.

The input for a %x format specifier is interpreted as a hexadecimal number.

The scanf() function scans each input field character by character. It might stop reading a particular input field either before it reaches a space character, when the specified width is reached, or when the next character cannot be converted as specified. When a conflict occurs between the specification and the input character, the next input field begins at the first unread character. The conflicting character, if there was one, is considered unread and is the first character of the next input field or the first character in subsequent read operations on stdin.

For %lc and %ls, specifies the data that is read is a multibyte string and is converted to wide characters as if by calls to mbtowc. Unless stdin has been overridden, the data is assumed to be in the CCSID of the job, and is converted to either wide EBCDIC, or if the source is compiled with LOCALETYPE(*LOCALEUCS2), into UNICODE.

Alternative format specification has the following form:



As an alternative, specific entries in the argument-list may be assigned by using the format specification outlined in the diagram above. This format specification and the previous format specification may not be mixed in the same call to scanf(). Otherwise, unpredictable results may occur.

The arg-number is a positive integer constant where 1 refers to the first entry in the argument-list. Arg-number may not be greater than the number of entries in the argument-list, or else the results are undefined. Arg-number also may not be greater than NL_ARGMAX.

Return Value

The scanf() function returns the number of fields that were successfully converted and assigned. The return value does not include fields that were read but not assigned.

The return value is EOF for an attempt to read at end-of-file if no conversion was performed. A return value of 0 means that no fields were assigned.

Error Conditions

If the type of the argument that is to be assigned into is different than the format specification, unpredictable results can occur. For example, reading a floating point value, but assigning it into a variable of type int, is incorrect and would have unpredictable results.

If there are more arguments than format specifications, the extra arguments are ignored. The results are undefined if there are not enough arguments for the format specifications.

If the format string contains an invalid format specification, and positional format specifications are being used, errno will be set to EILSEQ.

If positional format specifications are used and there are not enough arguments, errno will be set to EINVAL.

Examples using scanf()

This example scans various types of data.

```
#include <stdio.h>
int main(void)
  int i;
  float fp;
  char c, s[81];
  printf("Enter an integer, a real number, a character "
  "and a string : \n");
if (scanf("%d %f %c %s", &i, &fp, &c, s) != 4)
     printf("Not all fields were assigned\n");
  else
  {
     printf("integer = %d\n", i);
     printf("real number = %f\n", fp);
     printf("character = %c\n", c);
     printf("string = %s\n",s);
/******** If input is: 12 2.5 a yes, ************
****** then output should be similar to: ********
Enter an integer, a real number, a character and a string :
integer = 12
real number = 2.500000
character = a
string = yes
```

This example converts a hexadecimal integer to a decimal integer. The while loop ends if the input value is not a hexadecimal integer.

```
#include <stdio.h>
int main(void)
  int number;
  printf("Enter a hexadecimal number or anything else to quit:\n");
  while (scanf("%x",&number))
     printf("Hexadecimal Number = %x\n",number);
     printf("Decimal Number = %d\n", number);
}
/******* If input is: 0x231 0xf5e 0x1 q, *********
                                                                  **
****** then output should be similar to: ********
Enter a hexadecimal number or anything else to quit:
Hexadecimal Number = 231
Decimal Number = 561
Hexadecimal Number = f5e
Decimal Number = 3934
Hexadecimal Number = 1
Decimal Number
*/
```

This example reads from stdin and assigns data by using the alternative positional format string.

```
#include <stdio.h>
int main(int argc, char *argv[])
{
  int i;
  char s[20];
  float f;

scanf("%2$s %3$f %1$d",&i, s, &f);

printf("The data read was %i\n%s\n%f\n,i,s,f);

return 0;
  }

/* If the input is : test 0.2 100
    then the output will be similar to: */
  The data read was
100
  test
0.20000
```

This example reads in a multibyte character string into a wide UNICODE string. Compile with LOCALETYPE(*LOCALEUCS2)

```
#include <locale.h>
#include <stdio.h>
#include <wchar.h>

void main(void)
{
wchar_t uString[20];
setlocale(LC_UCS2_ALL, "");
scanf("Enter a string %1s",uString);
printf("String read was %1s\n",uString);
}
/* if the input is : ABC
    then the output will be similiar to:
    String read was ABC

*/
```

Related Information

- "fscanf() Read Formatted Data" on page 113
- "printf() Print Formatted Characters" on page 200
- "sscanf() Read Data" on page 322
- "<stdio.h>" on page 15
- "wscanf() Read Data Using Wide-Character Format String" on page 448
- "fwscanf() Read Data from Stream Using Wide Character" on page 128
- "swscanf() Read Wide Character Data" on page 369

setbuf() — Control Buffering

Format

```
#include <stdio.h>
void setbuf(FILE *, char *buffer);
```

Language Level: ANSI

Description

If the operating system supports user-defined buffers, setbuf() controls buffering for the specified stream. The setbuf() function only works in ILE C when using the integrated file system. The stream pointer must refer to an open file before any I/O or repositioning has been done.

If the buffer argument is NULL, the stream is unbuffered. If not, the buffer must point to a character array of length BUFSIZ, which is the buffer size that is defined in the <stdio.h> include file. The system uses the buffer, which you specify, for input/output buffering instead of the default system-allocated buffer for the given stream. stdout, stderr, and stdin do not support user-defined buffers.

The setvbuf() function is more flexible than the setbuf() function.

Note: Under ILE C, streams are fully buffered by default, with the exceptions of the stderr function, which is line-buffered, and memory files, which are unbuffered.

Return Value

There is no return value.

Example that uses setbuf()

This example opens the file setbuf.dat for writing. It then calls the setbuf() function to establish a buffer of length BUFSIZ. When string is written to the stream, the buffer buf is used and contains the string before it is flushed to the file.

```
#include <stdio.h>
int main(void)
  char buf[BUFSIZ];
   char string[] = "hello world";
  FILE *stream;
  memset(buf,'\0',BUFSIZ); /* initialize buf to null characters */
  stream = fopen("setbuf.dat", "wb");
  setbuf(stream,buf);
                            /* set up buffer */
   fwrite(string, sizeof(string), 1, stream);
  printf("%s\n",buf);
                          /* string is found in buf now */
   fclose(stream);
                           /* buffer is flushed out to myfile.dat */
```

Related Information

- "fclose() Close Stream" on page 75
- "fflush() Write Buffer to File" on page 80
- "fopen() Open Files" on page 92

- "setvbuf() Control Buffering" on page 311
- "<stdio.h>" on page 15

setjmp() — Preserve Environment

Format

```
#include <setjmp.h>
int setjmp(jmp_buf env);
```

Language Level: ANSI

Description

The setjmp() function saves a stack environment that can subsequently be restored by the longjmp() function. The setjmp() and longjmp() functions provide a way to perform a non-local goto. They are often used in signal handlers.

A call to the setjmp() function causes it to save the current stack environment in <code>env</code>. A subsequent call to the <code>longjmp()</code> function restores the saved environment and returns control to a point corresponding to the <code>setjmp()</code> call. The values of all variables (except register variables) accessible to the function receiving control contain the values they had when the <code>longjmp()</code> function was called. The values of register variables are unpredictable. Nonvolatile auto variables that are changed between calls to the <code>setjmp()</code> function and the <code>longjmp()</code> function are also unpredictable.

Return Value

The setjmp() function returns the value 0 after saving the stack environment. If the setjmp() function returns as a result of a longjmp() call, it returns the *value* argument of the longjmp() function, or 1 if the *value* argument of the longjmp() function is 0. There is no error return value.

Example that uses setjmp()

This example stores the stack environment at the statement if(setjmp(mark) != 0) ...

When the system first performs the if statement, it saves the environment in *mark* and sets the condition to FALSE because setjmp() returns a 0 when it saves the environment. The program prints the message:

```
setjmp has been called
```

The subsequent call to function p tests for a local error condition, which can cause it to perform the <code>longjmp()</code> function. Then, control returns to the original function using the environment that is saved in <code>mark</code>. This time, the condition is TRUE because -1 is the return value from the <code>longjmp()</code> function. The program then performs the statements in the block and prints:

```
setjmp has been called
```

Then the program calls the recover() function and exits.

```
#include <stdio.h>
#include <setjmp.h>
jmp buf mark;
```

```
void p(void);
void recover(void);
int main(void)
   if (setjmp(mark) != 0)
      printf("longjmp has been called\n");
      recover();
   printf("setjmp has been called\n");
  p();
   exit(1);
void p(void)
      longjmp(mark, -1);
void recover(void)
exit(1);
```

- "longjmp() Restore Stack Environment" on page 168
- "<setjmp.h>" on page 13

setlocale() — Set Locale

Format

#include <locale.h> char *setlocale(int category, const char *locale);

Language Level: ANSI

Thread Safe: NO.

Description

The setlocale() function changes or queries variables that are defined in the <locale.h> include file, that indicate location. The values for category are listed below.

Category	Purpose
LC_ALL	Names entire locale of program.
LC_COLLATE	Affects behavior of the strcoll() and strxfrm()functions.
LC_CTYPE	Affects behavior of character handling functions.
LC_MONETARY	Affects monetary information returned by localeconv() and nl_langinfo() functions.
LC_NUMERIC	Affects the decimal-point character for the formatted input/output and string conversion function, and the non-monetary formatting information returned by the localeconv() and nl_langinfo() functions.

Category	Purpose	
LC_TIME	Affects behavior of the strftime() function and the time formatting information returned by the nl_langinfo() function.	
LC_UCS2_ALL*	This category causes setlocale() to load the LC_*, LC_UCS2_CTYPE and the LC_UCS2_COLLATE categories from the locale specified. This category only accepts a locale with a UCS2 CCSID. LC_UCS2_ALL is identical to LC_ALL in function when LOCALETYPE(*LOCALEUCS2) is used.	
LC_UCS2_CTYPE*	This category causes the UCS2 CTYPE information to be set from the specified locale. The locale must have a UCS2 CCSID.	
LC_UCS2_COLLATE*	This category causes the UCS2 Collating sequence information to be set from the specified locale. The locale must have a UCS2 CCSID. Note: This category is not supported.	
LC_TOD	Affects the behavior of the time functions.	
	The category LC_TOD has several fields in it. The TNAME field is the time zone name. The TZDIFF field is the difference between local time and Greenwich Meridian time. If the TNAME field is nonblank, then the TZDIFF field is used when determining the values that are returned by some of the time functions. This value takes precedence over the system value, QUTCOFFSET.	
LC_MESSAGES	Affects the behavior of the catclose(), catgets() and catopen() functions and the message formatting information returned by the nl_langinfo() function.	
* For categories with UCS2 in the name, and for crtcmod or crtbndc, the option is LOCALETYPE(*LOCALEUCS2).		

Note: There are two ways of defining setlocale() and other locale-sensitive C functions on the iSeries server. The original way to define setlocale() uses *CLD locale objects to set the locale and retrieve locale-sensitive data. The second way to define setlocale() uses *LOCALE objects to set the locale and retrieve locale-sensitive data. The original way is accessed by specifying LOCALETYPE(*CLD) on the compilation command. The second way is accessed by specifying LOCALETYPE(*LOCALE) on the compilation command. For more information on the two methods of locale definition in ILE C, see "International Locale Support" in the WebSphere Development Studio: ILE C/C++ Programmer's Guide.

Setlocale using *CLD locale objects

You can set the value of *locale* to "C", "", LC_C, LC_C_GERMANY, LC_C_FRANCE, LC_C_SPAIN, LC_C_ITALY, LC_C_USA or LC_C_UK. A *locale* value of "C" indicates the default C environment. A *locale* value of "" tells the setlocale() function to use the default locale for the implementation.

Setlocale with *LOCALE objects.

You can set the value of *locale* to "", "C", "POSIX" or the fully qualified Integrated File System path name of a *LOCALE object enclosed in double quotes. A *locale* value of "C" or "POSIX" indicates the default C *LOCALE object. A *locale* value of "" tells the setlocale() function to use the default locale for the process.

If compiled with LOCALETYPE(*LOCALEUCS2), the locale must be a pointer to a valid UNICODE locale for the categories starting with LC_UCS2, and must not be a UNICODE locale for the other categories.

Return Value

The setlocale() function returns a pointer to a string that, if passed back to the setlocale() function, would restore the locale to the previous setting. This string should be copied by the user as it will be overwritten on subsequent calls to setlocale().

Note: Because the string to which a successful call to setlocale() points may be overwritten by subsequent calls to the setlocale() function, you should copy the string if you plan to use it later. The exact format of the locale string is different between locale types of *CLD, *LOCALE and *LOCALEUCS2.

To query the locale, give a NULL as the second parameter. For example, to query all the categories of your locale, enter the following statement:

```
char *string = setlocale(LC ALL, NULL);
```

Error Conditions

On error, the setlocale() function returns NULL, and the program's locale is not changed.

Example that uses *CLD locale objects

```
This example sets the locale of the program to
LC_C_FRANCE *CLD and prints the string
that is associated with the locale. This example must be compiled with
the LOCALETYPE(*CLD) parameter on the compilation command.
#include <stdio.h>
#include <locale.h>
char *string;
int main(void)
  string = setlocale(LC ALL, LC C FRANCE);
  if (string != NULL)
    printf(" %s \n", string);
```

Example that uses *LOCALE objects

/***********************************

This example sets the locale of the program to be "POSIX" and prints the string that is associated with the locale. This example must be compiled with the LOCALETYPE(*LOCALE) parameter on the CRTCMOD or CRTBNDC command.

```
#include <stdio.h>
#include <locale.h>
char *string;
int main(void)
   string = setlocale(LC ALL, "POSIX");
   if (string != NULL)
  printf(" %s \n", string);
}
```

Related Information

- "getenv() Search for Environment Variables" on page 135
- "localeconv() Retrieve Information from the Environment" on page 158
- "nl_langinfo() —Retrieve Locale Information" on page 196
- "<locale.h>" on page 7

setvbuf() — Control Buffering

Format

```
#include <stdio.h>
int setvbuf(FILE *stream, char *buf, int type, size_t size);
```

Language Level: ANSI

Thread Safe: YES.

Description

The setvbuf() function allows control over the buffering strategy and buffer size for a specified stream. The setvbuf() function only works in ILE C when using the integrated file system. The stream must refer to a file that has been opened, but not read or written to.

The array pointed to by buf designates an area that you provide that the C library may choose to use as a buffer for the stream. A buf value of NULL indicates that no such area is supplied and that the C library is to assume responsibility for managing its own buffers for the stream. If you supply a buffer, it must exist until the stream is closed.

The *type* must be one of the following:

Value Meaning

IONBF

No buffer is used.

_IOFBF

Full buffering is used for input and output. Use buf as the buffer and size as the size of the buffer.

_IOLBF

Line buffering is used. The buffer is deleted when a new-line character is written, when the buffer is full, or when input is requested.

If type is _IOFBF or _IOLBF, size is the size of the supplied buffer. If buf is NULL, the C library takes *size* as the suggested size for its own buffer. If *type* is _IONBF, both buf and size are ignored.

The value for *size* must be greater than 0.

Return Value

The setvbuf() function returns 0 if successful. It returns nonzero if a value that is not valid was specified in the parameter list, or if the request cannot be performed.

The setvbuf() function has no effect on stdout, stdin, or stderr.

Warning: The array that is used as the buffer must still exist when the specified stream is closed. For example, if the buffer is declared within the scope of a function block, the stream must be closed before the function is ended and frees the storage allocated to the buffer.

Example that uses setvbuf()

This example sets up a buffer of buf for stream1 and specifies that input to stream2 is to be unbuffered.

```
#include <stdio.h>
#define BUF SIZE 1024
char buf[BUF SIZE];
FILE *stream1, *stream2;
int main(void)
  stream1 = fopen("myfile1.dat", "r");
  stream2 = fopen("myfile2.dat", "r");
   /* stream1 uses a user-assigned buffer of BUF SIZE bytes */
  if (setvbuf(stream1, buf, _IOFBF, sizeof(buf)) != 0)
     printf("Incorrect type or size of buffer\n");
   /* stream2 is unbuffered
                                                             */
   if (setvbuf(stream2, NULL, _IONBF, 0) != 0)
     printf("Incorrect type or size of buffer\n");
/* This is a program fragment and not a complete function example */
```

Related Information

- "fclose() Close Stream" on page 75
- "fflush() Write Buffer to File" on page 80
- "fopen() Open Files" on page 92

signal() — Handle Interrupt Signals

Format

```
#include <signal.h>
void ( *signal (int sig, void(*func)(int)) )(int);
```

Language Level: ANSI

Description

ı

1

| | The signal() function allows a program to choose one of several ways to handle an interrupt signal from the operating system or from the raise() function. If compiled with the SYSIFCOPT(*ASYNCSIGNAL) option, this function uses asynchronous signals. The asynchronous version of this function behaves like sigaction() with SA_NODEFER and SA_RESETHAND options. Asynchronous signal handlers may not call abort() or exit(). The remainder of this function description will describe synchronous signals.

The *sig* argument must be one of the macros SIGABRT, SIGALL, SIGILL, SIGINT, SIGFPE, SIGIO, SIGOTHER, SIGSEGV, SIGTERM, SIGUSR1, or SIGUSR2, defined in the signal.h include file. SIGALL, SIGIO, and SIGOTHER are only supported by the ILE C/C++ Run-time library. The *func* argument must be one of the macros SIG_DFL or SIG_IGN, defined in the <signal.h> include file, or a function address.

The meaning of the values of *sig* is as follows:

Value Meaning

SIGABRT

Abnormal termination

SIGALL

Catch-all for signals whose current handling action is SIG DFL.

When SYSIFCOPT(*ASYNCSIGNAL) is specified, SIGALL is not a catch-all signal. A signal handler for SIGALL is only invoked for a user-raised SIGALL signal.

SIGILL

Detection of a function image that was not valid

SIGFPE

Arithmetic exceptions that are not masked, such as overflow, division by zero, and operations that are not valid

SIGINT

Interactive attention

SIGIO

Record file I/O error

SIGOTHER

ILE C signal

SIGSEGV

Access to memory that was not valid

SIGTERM

End request sent to the program

SIGUSR1

Intended for use by user applications. (extension to ANSI)

SIGUSR2

Intended for use by user applications. (extension to ANSI)

The action that is taken when the interrupt signal is received depends on the value of func.

Value Meaning

SIG_DFL

Default handling for the signal will occur.

SIG_IGN

The signal is to be ignored.

Return Value

A return value of SIG_ERR indicates an error in the call to signal(). If successful, the call to signal () returns the most recent value of func. The value of errno may be set to EINVAL (the signal is not valid).

Example that uses signal()

This example shows you how to establish a signal handler.

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#define ONE_K 1024
#define OUT OF STORAGE
                             (SIGUSR1)
/* The SIGNAL macro does a signal() checking the return code */
#define SIGNAL(SIG, StrCln)
  if (signal((SIG), (StrCln)) == SIG ERR) {
   perror("Could not signal user signal");
    abort();
void StrCln(int);
void DoWork(char **, int);
int main(int argc, char *argv[]) {
  int size;
  char *buffer;
  SIGNAL(OUT_OF_STORAGE, StrCln);
  if (argc != 2) {
   printf("Syntax: %s size \n", argv[0]);
    return(-1);
  size = atoi(argv[1]);
  DoWork(&buffer, size);
  return(0);
void StrCln(int SIG_TYPE) {
  printf("Failed trying to malloc storage\n");
  SIGNAL(SIG_TYPE, SIG_DFL);
  exit(0);
void DoWork(char **buffer, int size) {
  int rc;
  *buffer = malloc(size*ONE K);
                                   /* get the size in number of K */
  if (*buffer == NULL) {
     if (raise(OUT OF STORAGE)) {
       perror("Could not raise user signal");
        abort();
     }
  return;
/* This is a program fragment and not a complete function example */
Related Information

    "abort() — Stop a Program" on page 37

    "atexit() — Record Program Ending Function" on page 46

• "exit() — End Program" on page 73
• "raise() — Send Signal" on page 226
"<signal.h>" on page 14
```

signal() API in the API Reference Unix-type APIs.

sin() — Calculate Sine

Format

```
#include <math.h>
double sin(double x);
```

Description

The sin() function calculates the sine of x, with x expressed in radians. If x is too large, a partial loss of significance in the result may occur.

Return Value

The sin() function returns the value of the sine of x. The value of errno may be set to either EDOM or ERANGE.

Example that uses sin()

This example computes y as the sine of $\pi/2$.

```
#include <math.h>
#include <stdio.h>
int main(void)
  double pi, x, y;
  pi = 3.1415926535;
  x = pi/2;
  y = sin(x);
  printf("sin(%lf) = %lf\n", x, y);
/****************** Output should be similar to: ********
sin(1.570796) = 1.000000
```

Related Information

- "acos() Calculate Arccosine" on page 39
- "asin() Calculate Arcsine" on page 43
- "atan() atan2() Calculate Arctangent" on page 45
- "cos() Calculate Cosine" on page 64
- "cosh() Calculate Hyperbolic Cosine" on page 65
- "sinh() Calculate Hyperbolic Sine"
- "tan() Calculate Tangent" on page 371
- "tanh() Calculate Hyperbolic Tangent" on page 372
- "<math.h>" on page 8

sinh() — Calculate Hyperbolic Sine

Format

```
#include <math.h>
double sinh(double x);
```

Language Level: ANSI

Description

The sinh() function calculates the hyperbolic sine of x, with x expressed in radians.

Return Value

The sinh() function returns the value of the hyperbolic sine of x. If the result is too large, the sinh() function sets errno to ERANGE and returns the value HUGE_VAL (positive or negative, depending on the value of *x*).

Example that uses sinh()

This example computes y as the hyperbolic sine of $\pi/2$.

```
#include <math.h>
#include <stdio.h>
int main(void)
  double pi, x, y;
  pi = 3.1415926535;
  x = pi/2;
  y = sinh(x);
  printf("sinh(%lf) = %lf\n", x, y);
/******* Output should be similar to: ********
sinh(1.570796) = 2.301299
```

Related Information

- "acos() Calculate Arccosine" on page 39
- "asin() Calculate Arcsine" on page 43
- "atan() atan2() Calculate Arctangent" on page 45
- "cos() Calculate Cosine" on page 64
- "cosh() Calculate Hyperbolic Cosine" on page 65
- "sin() Calculate Sine" on page 315
- "tan() Calculate Tangent" on page 371
- "tanh() Calculate Hyperbolic Tangent" on page 372
- "<math.h>" on page 8

sprintf() — Print Formatted Data to Buffer

Format

```
#include <stdio.h>
int sprintf(char *buffer, const char *format-string, argument-list);
```

Language Level: ANSI

Description

The sprintf() function formats and stores a series of characters and values in the array buffer. Any argument-list is converted and put out according to the corresponding format specification in the format-string.

The format-string consists of ordinary characters and has the same form and function as the *format-string* argument for the printf() function.

Return Value

The sprintf() function returns the number of bytes that are written in the array, not counting the ending null character.

Example that uses sprintf()

This example uses sprintf() to format and print various data.

```
#include <stdio.h>
char buffer[200];
int i, j;
double fp;
char *s = "baltimore";
char c;
int main(void)
   c = '1';
   i = 35;
   fp = 1.7320508;
   /* Format and print various data */
   j = sprintf(buffer, "%s\n", s);
  j += sprintf(buffer+j, "%c\n", c);
j += sprintf(buffer+j, "%d\n", i);
j += sprintf(buffer+j, "%f\n", fp);
   printf("string:\n%s\ncharacter count = %d\n", buffer, j);
/******* Output should be similar to: ********
string:
baltimore
35
1.732051
character count = 24
                                   */
```

Related Information

- "fprintf() Write Formatted Data to a Stream" on page 99
- "printf() Print Formatted Characters" on page 200
- "sscanf() Read Data" on page 322
- "swprintf() Format and Write Wide Characters to Buffer" on page 367
- "vfprintf() Print Argument Data to Stream" on page 386
- "vprintf() Print Argument Data" on page 389
- "vsprintf() Print Argument Data to Buffer" on page 391
- "<stdio.h>" on page 15

sqrt() — Calculate Square Root

Format

```
#include <math.h>
double sqrt(double x);
```

Description

The sqrt() function calculates the nonnegative value of the square root of x.

Return Value

The sqrt() function returns the square root result. If x is negative, the function sets errno to EDOM, and returns 0.

Example that uses sqrt()

This example computes the square root of the quantity that is passed as the first argument to main. It prints an error message if you pass a negative value.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main(int argc, char ** argv)
 char * rest;
 double value;
 if ( argc != 2 )
   printf( "Usage: %s value\n", argv[0] );
 else
   value = strtod( argv[1], &rest);
   if ( value < 0.0 )
      printf( "sqrt of a negative number\n" );
      printf("sqrt( %lf ) = %lf\n", value, sqrt( value ));
/************** If the input is 45, *******************
******* then the output should be similar to: *******
sqrt( 45.000000 ) = 6.708204
```

Related Information

- "exp() Calculate Exponential Function" on page 73
- "hypot() Calculate Hypotenuse" on page 145
- "log() Calculate Natural Logarithm" on page 165
- "log10() Calculate Base 10 Logarithm" on page 166
- "pow() Compute Power" on page 199
- "<math.h>" on page 8

srand() — Set Seed for rand() Function

Format

```
#include <stdlib.h>
void srand(unsigned int seed);
```

Thread Safe: NO.

Description

The srand() function sets the starting point for producing a series of pseudo-random integers. If srand() is not called, the rand() seed is set as if srand(1) were called at program start. Any other value for *seed* sets the generator to a different starting point.

The rand() function generates the pseudo-random numbers.

Return Value

There is no return value.

Example that uses srand()

This example first calls srand() with a value other than 1 to initiate the random value sequence. Then the program computes five random values for the array of integers that are called **ranvals**.

Related Information

- "rand(), rand_r() Generate Random Number" on page 227
- "<stdlib.h>" on page 16

snprintf() — Print Formatted Data to Buffer

Description

ı

I

Ι

The snprintf() function formats and stores a series of characters and values in the array *buffer*. Any *argument-list* is converted and put out according to the corresponding format specification in the *format-string*. The snprintf() function is identical to the sprintf() function with the addition of the *n* argument, which indicates the maximum number of characters (including the ending null character) to be written to *buffer*.

The *format-string* consists of ordinary characters and has the same form and function as the format string for the printf() function.

Return Value

The snprintf() function returns the number of bytes that are written in the array, not counting the ending null character.

Example that uses snprintf()

This example uses snprintf() to format and print various data.

```
#include <stdio.h>
char buffer[200];
int i, j;
double fp;
char *s = "baltimore";
char c;
int main(void)
   c = '1';
   i = 35;
   fp = 1.7320508;
   /* Format and print various data */
   j = sprintf(buffer, 6, "%s\n", s);
   j += sprintf(buffer+j, 6, "%c\n", c);
j += sprintf(buffer+j, 6, "%d\n", i);
   j += sprintf(buffer+j, 6, "%f\n", fp);
   printf("string:\n%s\ncharacter count = %d\n", buffer, j);
/******************* Output should be similar to: *********
string:
baltil
35
1.732
character count = 15
                                 */
```

Related Information

- "fprintf() Write Formatted Data to a Stream" on page 99
- "printf() Print Formatted Characters" on page 200
- "sprintf() Print Formatted Data to Buffer" on page 317
- "vsnprintf() Print Argument Data to Buffer" on page 390
- "<stdio.h>" on page 15

sscanf() — Read Data

Format

```
#include <stdio.h>
int sscanf(const char *buffer, const char *format, argument-list);
```

Language Level: ANSI

Description

The sscanf() function reads data from buffer into the locations that are given by argument-list. Each argument must be a pointer to a variable with a type that corresponds to a type specifier in the format-string.

Return Value

The sscanf() function returns the number of fields that were successfully converted and assigned. The return value does not include fields that were read but not assigned.

The return value is EOF when the end of the string is encountered before anything is converted.

Example that uses sscanf()

This example uses sscanf() to read various data from the string tokenstring, and then displays that data.

```
#include <stdio.h>
#include <stddef.h>
int main(void)
           *tokenstring = "15 12 14";
   wchar t *widestring = L"ABC Z";
   wchar_t ws[81];
   wchar_t wc;
   int i;
    float fp;
   char s[81];
    char c;
   /* Input various data
    /* In the first invocation of sscanf, the format string is
    /* "%s %c%d%f". If there were no space between %s and %c,
    /* sscanf would read the first character following the
    /* string, which is a blank space.
    sscanf(tokenstring, "%s %c%d%f", s, &c, &i, &fp);
    sscanf((char *)widestring, "%S %C", ws,&wc);
    /* Display the data */
    printf("\nstring = %s\n",s);
    printf("character = %c\n",c);
   printf("integer = %d\n",i);
    printf("floating-point number = %f\n",fp);
   printf("wide-character string = %S\n",ws);
   printf("wide-character = %C\n",wc);
```

/******* Output should be similar to: *********

Related Information

- "fscanf() Read Formatted Data" on page 113
- "scanf() Read Data" on page 299
- "swscanf() Read Wide Character Data" on page 369
- "fwscanf() Read Data from Stream Using Wide Character" on page 128
- "wscanf() Read Data Using Wide-Character Format String" on page 448
- "sprintf() Print Formatted Data to Buffer" on page 317
- "<stdio.h>" on page 15

strcasecmp() — Compare Strings without Case Sensitivity

Format

```
#include <strings.h>
int srtcasecmp(const char *string1, const char *string2);
```

Language Level: XPG4

Description

The strcasecmp() function compares *string1* and *string2* without sensitivity to case. All alphabetic characters in *string1* and *string2* are converted to lowercase before comparison.

The strcasecmp() function operates on null terminated strings. The string arguments to the function are expected to contain a null character (' $\0$ ') marking the end of the string.

Return Value

Thestrcasecmp() function returns a value indicating the relationship between the two strings, as follows:

Table 6. Return values of strcasecmp()

Value	Meaning
Less than 0	string1 less than string2
0	string1 equivalent to string2
Greater than 0	string1 greater than string2

Example that usesstrcasecmp()

This example uses strcasecmp() to compare two strings.

```
#include <stdio.h>
#include <strings.h>
int main(void)
 char t *str1 = "STRING";
 char t *str2 = "string";
 int result;
 result = strcasecmp(str1, str2);
 if (result == 0)
   printf("Strings compared equal.\n");
 else if (result < 0)
   printf("\"%s\" is less than \"%s\".\n", str1, str2);
   printf("\"%s\" is greater than \"%s\".\n", str1, str2);
 return 0;
/***** The output should be similar to: *******
Strings compared equal.
************
```

Related Information

- "strncasecmp() Compare Strings without Case Sensitivity" on page 341
- "strncmp() Compare Strings" on page 344
- "stricmp Compare Strings without Case Sensitivity" on page 340
- "wcscmp() Compare Wide-Character Strings" on page 403
- "wcsncmp() Compare Wide-Character Strings" on page 411
- "_wcsicmp Compare Wide Character Strings without Case Sensitivity" on page 413
- "_wcsnicmp Compare Wide Character Strings without Case Sensitivity" on page 415
- "<strings.h>" on page 17

strcat() — Concatenate Strings

Format

```
#include <string.h>
char *strcat(char *string1, const char *string2);
```

Language Level: ANSI

Description

The strcat() function concatenates *string2* to *string1* and ends the resulting string with the null character.

The strcat() function operates on null-ended strings. The string arguments to the function should contain a null character (\0) that marks the end of the string. No length checking is performed. You should not use a literal string for a string1 value, although *string2* may be a literal string.

If the storage of *string1* overlaps the storage of *string2*, the behavior is undefined.

Return Value

The strcat() function returns a pointer to the concatenated string (string1).

Example that uses strcat()

This example creates the string "computer program" using strcat().

```
#include <stdio.h>
#include <string.h>
#define SIZE 40
int main(void)
 char buffer1[SIZE] = "computer";
 char * ptr;
 ptr = strcat( buffer1, " program" );
 printf( "buffer1 = %s\n", buffer1 );
/****** Output should be similar to: *********
buffer1 = computer program
```

Related Information

- "strchr() Search for Character"
- "strcmp() Compare Strings" on page 326
- "strcpy() Copy Strings" on page 330
- "strcspn() Find Offset of First Character Match" on page 331
- "strncat() Concatenate Strings" on page 343
- "wcscat() Concatenate Wide-Character Strings" on page 400
- "wcsncat() Concatenate Wide-Character Strings" on page 410
- "<string.h>" on page 17

strchr() — Search for Character

Format

```
#include <string.h>
char *strchr(const char *string, int c);
```

Language Level: ANSI

Description

The strchr() function finds the first occurrence of a character in a string. The character *c* can be the null character (\0); the ending null character of *string* is included in the search.

The strchr() function operates on null-ended strings. The string arguments to the function should contain a null character (\0) that marks the end of the string.

Return Value

The strchr() function returns a pointer to the first occurrence of c that is converted to a character in string. The function returns NULL if the specified character is not found.

Example that uses strchr()

This example finds the first occurrence of the character "p" in "computer program".

```
#include <stdio.h>
#include <string.h>
#define SIZE 40
int main(void)
 char buffer1[SIZE] = "computer program";
 char * ptr;
 int ch = 'p';
 ptr = strchr( buffer1, ch );
 printf( "The first occurrence of %c in '%s' is '%s'\n",
           ch, buffer1, ptr );
/****** Output should be similar to: *********
The first occurrence of p in 'computer program' is 'puter program'
```

Related Information

- "strcat() Concatenate Strings" on page 324
- "strcmp() Compare Strings"
- "strcpy() Copy Strings" on page 330
- "strcspn() Find Offset of First Character Match" on page 331
- "strncmp() Compare Strings" on page 344
- "strpbrk() Find Characters in String" on page 348
- "strrchr() Locate Last Occurrence of Character in String" on page 354
- "strspn() —Find Offset of First Non-matching Character" on page 355
- "wcschr() Search for Wide Character" on page 402
- "wcsspn() Find Offset of First Non-matching Wide Character" on page 420
- "<string.h>" on page 17

strcmp() — Compare Strings

Format

```
#include <string.h>
int strcmp(const char *string1, const char *string2);
```

Language Level: ANSI

Description

The strcmp() function compares *string1* and *string2*. The function operates on null-ended strings. The string arguments to the function should contain a null character (\0) that marks the end of the string.

Return Value

The strcmp() function returns a value indicating the relationship between the two strings, as follows:

Value	Meaning
Less than 0	string1 less than string2
0	string1 identical to string2
Greater than 0	string1 greater than string2

Example that uses strcmp()

This example compares the two strings that are passed to main() using strcmp().

```
#include <stdio.h>
#include <string.h>
int main(int argc, char ** argv)
 int result;
 if ( argc != 3 )
  printf( "Usage: %s string1 string2\n", argv[0] );
 else
  result = strcmp( argv[1], argv[2] );
  if ( result == 0 )
    printf( "\"s\" is identical to \"s\"\n", argv[1], argv[2] );
  else if ( result < 0 )
    printf( "\"%s\" is less than \"%s\"\n", argv[1], argv[2] );
    printf( "\"s\" is greater than \"s\"\n", argv[1], argv[2] );
****** "is this first?" and "is this before that one?", *******
****** then the expected output is: *********
"is this first?" is greater than "is this before that one?"
```

Related Information

- "strcat() Concatenate Strings" on page 324
- "strchr() Search for Character" on page 325
- "strcpy() Copy Strings" on page 330
- "strcspn() Find Offset of First Character Match" on page 331
- "strncmp() Compare Strings" on page 344
- "strpbrk() Find Characters in String" on page 348
- "strrchr() Locate Last Occurrence of Character in String" on page 354
- "strspn() —Find Offset of First Non-matching Character" on page 355
- "wcschr() Search for Wide Character" on page 402
- "wcsspn() Find Offset of First Non-matching Wide Character" on page 420

strcmpi - Compare Strings Without Case Sensitivity

Format

```
#include <string.h>
int strcmpi(const char *string1, const char *string2);
```

Note: The strcmpi function is supported only for C++, not for C.

Language Level: Extension

Description

strcmpi compares *string1* and *string2* without sensitivity to case. All alphabetic characters in the two arguments *string1* and *string2* are converted to lowercase before the comparison.

The function operates on null-ended strings. The string arguments to the function are expected to contain a null character (\0) marking the end of the string.

Return Value

strcmpi returns a value indicating the relationship between the two strings, as follows:

Value	Meaning
Less than 0	string1 less than string2
0	string1 equivalent to string2
Greater than 0	string1 greater than string2

*/

Example that uses strcmpi()

This example uses strcmpi to compare two strings.

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    /* Compare two strings without regard to case
    if (0 == strcmpi("hello", "HELLO"))
        printf("The strings are equivalent.\n");
    else
        printf("The strings are not equivalent.\n");
    return 0;
}
```

The output should be:

The strings are equivalent.

Related Information:

- strcoll
- strcspn
- "strdup Duplicate String" on page 332
- "stricmp Compare Strings without Case Sensitivity" on page 340

- strncmp
- "strnicmp Compare Substrings Without Case Sensitivity" on page 347
- wcscmp
- wcsncmp
- <string.h>
- strcasecmp
- strncasecmp

strcoll() — Compare Strings

Format

```
#include <string.h>
int strcoll(const char *string1, const char *string2);
```

Language Level: ANSI

Description

The strcoll() function compares two strings using the collating sequence that is specified by the program's locale.

The behavior of this function is affected by the LC_COLLATE category of the current locale.

Return Value

The strcoll() function returns a value indicating the relationship between the strings, as listed below:

Value	Meaning
Less than 0	string1 less than string2
0	string1 equivalent to string2
Greater than 0	string1 greater than string2

If strcoll() is unsuccessful, errno is changed. The value of errno may be set to EINVAL (the *string1* or *string2* arguments contain characters that are not available in the current locale).

Example that uses strcoll()

This example compares the two strings that are passed to main() using strcoll():

```
#include <stdio.h>
#include <string.h>
int main(int argc, char ** argv)
 int result;
 if ( argc != 3 )
   printf( "Usage: %s string1 string2\n", argv[0] );
 else
   result = strcoll( argv[1], argv[2] );
   if (result == 0)
     printf( "\"s\" is identical to \"s\"\n", argv[1], argv[2] );
   else if ( result < 0 )
     printf( "\"%s\" is less than \"%s\"\n", argv[1], argv[2] );
     printf( "\"s\" is greater than \"s\"\n", argv[1], argv[2] );
/************* If the input is the strings *****************
****** "firststring" and "secondstring", ***********
****** then the expected output is: *********
"firststring" is less than "secondstring"
```

Related Information

- "setlocale() Set Locale" on page 308
- "strcmp() Compare Strings" on page 326
- "strncmp() Compare Strings" on page 344
- "wcscoll() —Language Collation String Comparison" on page 404
- "<string.h>" on page 17

strcpy() — Copy Strings

Format

```
#include <string.h>
char *strcpy(char *string1, const char *string2);
```

Language Level: ANSI

Description

The strcpy() function copies string2, including the ending null character, to the location that is specified by *string1*.

The strcpy() function operates on null-ended strings. The string arguments to the function should contain a null character (\0) that marks the end of the string. No length checking is performed. You should not use a literal string for a string1 value, although string2 may be a literal string.

Return Value

The strcpy() function returns a pointer to the copied string (*string1*).

Example that uses strcpy()

This example copies the contents of source to destination.

Related Information

- "strcat() Concatenate Strings" on page 324
- "strchr() Search for Character" on page 325
- "strcmp() Compare Strings" on page 326
- "strcspn() Find Offset of First Character Match"
- "strncpy() Copy Strings" on page 346
- "strpbrk() Find Characters in String" on page 348
- "strrchr() Locate Last Occurrence of Character in String" on page 354
- "strspn() —Find Offset of First Non-matching Character" on page 355
- "wcscpy() Copy Wide-Character Strings" on page 405
- "wcsncpy() Copy Wide-Character Strings" on page 413
- "<string.h>" on page 17

strcspn() — Find Offset of First Character Match

Format

```
#include <string.h>
size_t strcspn(const char *string1, const char *string2);
```

Language Level: ANSI

Description

The strcspn() function finds the first occurrence of a character in *string1* that belongs to the set of characters that is specified by *string2*. Null characters are not considered in the search.

The strcspn() function operates on null-ended strings. The string arguments to the function should contain a null character (\0) marking the end of the string.

Return Value

The strcspn() function returns the index of the first character found. This value is equivalent to the length of the initial substring of string1 that consists entirely of characters not in string2.

Example that uses strcspn()

This example uses strcspn() to find the first occurrence of any of the characters "a", "x", "l", or "e" in *string*.

```
#include <stdio.h>
#include <string.h>
#define SIZE 40
int main(void)
 char string[SIZE] = "This is the source string";
 char * substring = "axle";
 printf( "The first %i characters in the string \"%s\" "
         "are not in the string \"%s\" \n",
           strcspn(string, substring), string, substring);
/***** Output should be similar to: ********
The first 10 characters in the string "This is the source string"
are not in the string "axle"
```

Related Information

- "strcat() Concatenate Strings" on page 324
- "strchr() Search for Character" on page 325
- "strcmp() Compare Strings" on page 326
- "strcpy() Copy Strings" on page 330
- "strncmp() Compare Strings" on page 344
- "strpbrk() Find Characters in String" on page 348
- "strrchr() Locate Last Occurrence of Character in String" on page 354
- "strspn() —Find Offset of First Non-matching Character" on page 355
- "<string.h>" on page 17

strdup - Duplicate String

Format

```
#include <string.h>
char *strdup(const char *string);
```

Note: The strdup function is supported only for C++, not for C.

Language Level: XPG4, Extension

Description

strdup reserves storage space for a copy of *string* by calling malloc. The string argument to this function is expected to contain a null character (\0) marking the end of the string. Remember to free the storage reserved with the call to strdup.

Return Value

strdup returns a pointer to the storage space containing the copied string. If it cannot reserve storage strdup returns NULL.

Example that uses strdup()

This example uses strdup to duplicate a string and print the copy.

```
#include <stdio.h>
#include <string.h>
int main(void)
{
   char *string = "this is a copy";
   char *newstr;
   /* Make newstr point to a duplicate of string
   if ((newstr = strdup(string)) != NULL)
      printf("The new string is: %s\n", newstr);
   return 0;
}
```

The output should be:

The new string is: this is a copy

Related Information:

- strcpy
- strncpy
- wcscpy
- wcsncpy
- Strings
- <string.h>
- "wcscspn() Find Offset of First Wide-Character Match" on page 406

strerror() — Set Pointer to Run-Time Error Message

Format

```
#include <string.h>
char *strerror(int errnum);
```

Language Level: ANSI

Description

The strerror() function maps the error number in *errnum* to an error message string.

Return Value

The strerror() function returns a pointer to the string. The function returns the string in the CCSID of the job. The function does not use the program locale in any way. It does not return a NULL value.

Example that uses strerror()

This example opens a file and prints a run-time error message if an error occurs.

```
#include <stdlib.h>
#include <string.h>
#include <errno.h>
int main(void)
  FILE *stream;
  if ((stream = fopen("mylib/myfile", "r")) == NULL)
     printf(" %s \n", strerror(errno));
/* This is a program fragment and not a complete function example */
```

Related Information

- "clearerr() Reset Error Indicators" on page 62
- "ferror() Test for Read/Write Errors" on page 79
- "perror() Print Error Message" on page 198
- "<string.h>" on page 17

strfmon() — Convert Monetary Value to String

Format

```
#include <monetary.h>
int strfmon(char *s, size_t maxsize, const char *format, argument_list);
```

Language Level: XPG4

Thread Safe: YES.

Description

The strfmon() function places characters into the array pointed to by s as controlled by the string pointed to by format. No more than maxsize characters are placed into the array.

The character string format contains two types of objects: plain characters, which are copied to the output stream, and directives, each of which results in the fetching of zero or more arguments, which are converted and formatted. The results are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments are simply ignored. Only 15 significant digits are guaranteed on conversions involving double values.

A directive consists of a % character, optional conversion specifications, and a ending character that determines the directive's behavior.

A directive consists of the following sequence:

- A % character.
- · Optional flags.
- · Optional field width.
- · Optional left precision.
- Optional right precision.
- A required conversion character indicating the type of conversion to be performed.

Table 7. Flags

Flag	Meaning
=f	An = followed by a single character f which is used as the numeric fill character. By default the numeric fill character is a space character. This flag does not affect field width filling, which always uses a space character. This flag is ignored unless left precision is specified.
	Do not use grouping characters when formatting the currency value. Default is to insert grouping characters as defined in the current locale.
+ or (Specify the style representing positive and negative currency amounts. If + is specified, the locale's equivalent of + and – for monetary quantities will be used. If (is specified, negative amounts are enclosed within parenthesis. Default is +.
!	Do not output the currency symbol. Default is to output the currency symbol.
-	Use left justification for double arguments. Default is right justification.

Field Width

w A decimal digit string *w* specifying a minimum field width in bytes in which the result of the conversion is right-justified (or left-justified if the flag - is specified). The default is 0.

Left Precision

#n A # followed by a decimal digit string n specifying a maximum number of digits expected to be formatted to the left of the radix character. This option can be used to keep the formatted output from multiple calls to strfmon() aligned in the same columns. It can also be used to fill unused positions with a special character as in \$***123.45. This option causes an amount to be formatted as if it has the number of digits specified by n. If more than n digit positions are required, this conversion specification is ignored. Digit positions in excess of those actually required are filled with the numeric fill character (see the =f flag above).

If grouping has not been suppressed withe the flag, and it is defined for the current locale, grouping separators are inserted before the fill characters (if any) are added. Grouping separators are not applied to fill characters even if the fill character is a digit. To ensure alignment, any characters appearing before or after the number in the formatted output, such as currency or sign symbols, are padded as necessary with space characters to make their positive and negative formats an equal length.

Right Precision

.p A period followed by a decimal digit string *p* specifies the number of digits after the radix character. If the value of the right precision *p* is 0, no radix character appears. If a right precision is not specified, a default specified by the current locale is used. The amount being formatted is rounded to the specified number of digits prior to formatting.

Table 8. Conversion Characters

Specifier	Meaning
%i	The double argument is formatted according to the locale's international currency format.
%n	The double argument is formatted according to the locale's national currency format.
%%	Is replaced by %. No argument is converted.

Note: This function is only accessible if LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

Return Value

If the total number of resulting bytes including the ending null character is not more than *maxsize*, the strfmon() function returns the number of bytes placed into the array pointed to by s, but excludes the ending null character. Otherwise, zero is returned, and the contents of the array are undefined.

The value of errno may be set to:

E2BIG Conversion stopped due to lack of space in the buffer.

Example that uses strfmon()

```
#include <stdio.h>
#include <monetary.h>
#include <locale.h>
int main(void)
   char string[100];
   double money = 1234.56;
   if (setlocale(LC_ALL, "/qsys.lib/en_us.locale") == NULL) {
       printf("Unable to setlocale().\n");
       exit(1);
   strfmon(string, 100, "%i", money); /* USD 1,234.56 */
   printf("%s\n", string);
   strfmon(string, 100, "%n", money); /* $1,234.56 */
   printf("%s\n", string);
/**********************
        The output should be similar to:
        USD 1,234.56
        $1,234.56
*************************************
```

Related Information

- "<monetary.h>" on page 8
- "localeconv() Retrieve Information from the Environment" on page 158

strftime() — Convert Date/Time to String

```
Format
```

```
#include <time.h>
size t strftime(char *s, size t maxsize, const char *format,
                   const struct tm *timeptr);
```

Thread Safe: YES.

Description

I

The strfttime() function places bytes into the array pointed to by s as controlled by the string pointed to by format. The format string consists of zero or more conversion specifications and ordinary characters. A conversion specification consists of a % character and a terminating conversion character that determines the behavior of the conversion. All ordinary characters (including the terminating null byte, and multi-byte chars) are copied unchanged into the array. If copying takes place between objects that overlap, then the behavior is undefined. No more than maxsize bytes are placed in the array. The appropriate characters are determined by the values contained in the structure pointed to by timeptr, and by the values stored in the current locale.

Each standard conversion specification is replaced by appropriate characters as described in the following table:

Specifier	Meaning
%a	Abbreviated weekday name.
%A	Full weekday name.
%b	Abbreviated month name.
%B	Full month name.
%с	Date/Time in the format of the locale.
%C	Century number [00-99], the year divided by 100 and truncated to an integer.
%d	Day of the month [01-31].
%D	Date Format, same as %m/%d/%y.
%e	Same as %d, except single digit is preceded by a space [1-31].
%h	Same as %b.
%H	Hour in 24-hour format [00-23].
%I	Hour in 12-hour format [01-12].
%j	Day of the year [001-366].
%m	Month [01-12].
%M	Minute [00-59].
%n	Newline character.
%p	AM or PM string.
%r	Time in AM/PM format of the locale. If not available in the locale time format, defaults to the POSIX time AM/PM format: %I:%M:%S %p.
%R	24-hour time format without seconds, same as %H:%M.
%S	Second [00-61]. The range for seconds allows for a leap second and a double leap second.
%t	Tab character.
%T	24-hour time format with seconds, same as %H:%M:%S.
%u	Weekday [1,7]. Monday is 1 and Sunday is 7.
%U	Week number of the year [00-53]. Sunday is the first day of the week.

I	
I	
I	
I	
I	
I	
I	
I	
I	
I	
ı	

Specifier	Meaning
%V	ISO week number of the year[01-53]. Monday is the first day of the week. If the week containing January 1st has four or more days in the new year then it is considered week 1. Otherwise, it is the last week of the previous year, and the next year is week 1 of the new year.
%w	Weekday [0,6], Sunday is 0.
%W	Week number of the year [00-53]. Monday is the first day of the week.
%x	Date in the format of the locale.
%X	Time in the format of the locale.
%y	2 digit year [00,99].
%Y	4-digit year. Can be negative.
%Z	Time zone name. Available only from the locale.
%%	% character.

Modified Conversion Specifiers

Some conversion specifiers can be modified by the E or O modifier characters to indicate that an alternate format or specification should be used rather than the one normally used by the unmodified conversion specifier. If a modified conversion specifier uses a field in the current locale that is unavailable, then the behavior will be as if the unmodified conversion specification were used. For example, if the era string is the empty string "", which means that the string is unavailable, then %EY would act like %Y.

Specifier	Meaning
%Ec	Date/time for current era.
%EC	Era name.
%Ex	Date for current era.
%EX	Time for current era.
%Ey	Era year. This is the offset from the base year.
%EY	Year for current era.
%Od	Day of the month using alternate digits.
%Oe	Same as %Od.
%OH	Hour in 24 hour format using alternate digits.
%OI	Hour in 12 hour format using alternate digits.
%Om	Month using alternate digits.
%OM	Minutes using alternate digits.
%OS	Seconds using alternate digits.
%Ou	Weekday using alternate digits. Monday is 1 and Sunday is 7.
%OU	Week number of the year using alternate digits. Sunday is the first day of the week.
%OV	ISO week number of the year using alternate digits. See %V for explanation on ISO week number.
%Ow	Weekday using alternate digits. Sunday is 0.
%OW	Week number of the year using alternate digits. Monday is the first day of the week.

Specifier	Meaning
%Оу	2-digit year using alternate digits.

Note: %C, %D, %e, %h, %n, %r, %R, %t, %T, %u, %V and the modified conversion specifiers are not available unless LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

Return Value

| |

If the total number of resulting bytes including the terminating null byte is not more than maxsize, strftime() returns the number of bytes placed into the array pointed to by s, not including the terminating null byte. Otherwise, 0 is returned and the contents of the array are indeterminate.

Example that uses strftime()

```
#include <stdio.h>
#include <time.h>
int main(void)
    char s[100];
    int rc;
    time t temp;
    struct tm *timeptr;
    temp = time(NULL);
    timeptr = localtime(&temp);
    rc = strftime(s,sizeof(s),"Today is %A, %b %d.\nTime: %r", timeptr);
    printf("%d characters written.\n%s\n",rc,s);
    return 0;
/****************
   The output should be similar to:
   46 characters written
   Today is Wednesday, Oct 24.
   Time: 01:01:15 PM
```

Related Information

- "asctime() Convert Time to Character String" on page 40
- "asctime_r() Convert Time to Character String (Restartable)" on page 42
- "ctime() Convert Time to Character String" on page 66
- "ctime_r() Convert Time to Character String (Restartable)" on page 67
- "gmtime() Convert Time" on page 141
- "gmtime_r() Convert Time (Restartable)" on page 143
- "localtime() Convert Time" on page 163
- "localtime_r() Convert Time (Restartable)" on page 164
- "setlocale() Set Locale" on page 308
- "strptime()— Convert String to Date/Time" on page 350
- "time() Determine Current Time" on page 373
- "<time.h>" on page 17

stricmp - Compare Strings without Case Sensitivity

Format

```
#include <string.h>
int stricmp(const char *string1, const char *string2);
```

Note: The stricmp function is supported only for C++, not for C.

Language Level: Extension

Description

stricmp compares *string1* and *string2* without sensitivity to case. All alphabetic characters in the two arguments string1 and string2 are converted to lowercase before the comparison.

The function operates on null-ended strings. The string arguments to the function are expected to contain a null character (\0) marking the end of the string.

Return Value

stricmp returns a value indicating the relationship between the two strings, as follows:

Value	Meaning	
Less than 0	string1 less than string2	
0	string1 equivalent to string2	
Greater than 0	string1 greater than string2	

Example that uses stricmp()

This example uses stricmp to compare two strings.

```
#include <stdio.h>
#include <string.h>
int main(void)
   /* Compare two strings as lowercase
                                                                     */
  if (0 == stricmp("hello", "HELLO"))
     printf("The strings are equivalent.\n");
     printf("The strings are not equivalent.\n");
   return 0;
```

The output should be:

The strings are equivalent.

Related Information:

- "strcmpi Compare Strings Without Case Sensitivity" on page 328
- strcoll
- strcspn
- "strdup Duplicate String" on page 332
- "strcasecmp() Compare Strings without Case Sensitivity" on page 323

- "strncasecmp() Compare Strings without Case Sensitivity"
- "strnicmp Compare Substrings Without Case Sensitivity" on page 347
- wcscmp
- wcsncmp
- <string.h>

strlen() — Determine String Length

Format

```
#include <string.h>
size t strlen(const char *string);
```

Language Level: ANSI

Description

The strlen() function determines the length of *string* excluding the ending null character.

Return Value

The strlen() function returns the length of *string*.

Example that uses strlen()

This example determines the length of the string that is passed to main().

Related Information

- "mblen() Determine Length of a Multibyte Character" on page 172
- "strncat() Concatenate Strings" on page 343
- "strncmp() Compare Strings" on page 344
- "strncpy() Copy Strings" on page 346
- "wcslen() Calculate Length of Wide-Character String" on page 409
- "<string.h>" on page 17

strncasecmp() — Compare Strings without Case Sensitivity

Format

```
#include <strings.h>
int srtncasecmp(const char *string1, const char *string2, size t count);
```

Language Level: XPG4

Description

The strncasecmp() function compares up to *count* characters of *string1* and *string2* without sensitivity to case. All alphabetic characters in string1 and string2 are converted to lowercase before comparison.

The strncasecmp() function operates on null terminated strings. The string arguments to the function are expected to contain a null character ('\0') marking the end of the string.

Return Value

Thestrncasecmp() function returns a value indicating the relationship between the two strings, as follows:

Table 9. Return values of strncasecmp()

Value	Meaning
Less than 0	string1 less than string2
0	string1 equivalent to string2
Greater than 0	string1 greater than string2

Example that usesstrncasecmp()

```
This example uses strncasecmp() to compare two strings.
#include <stdio.h>
#include <strings.h>
int main(void)
 char t *str1 = "STRING ONE";
 char t *str2 = "string TWO";
 int result;
 result = strncasecmp(str1, str2, 6);
 if (result == 0)
   printf("Strings compared equal.\n");
 else if (result < 0)
   printf("\"%s\" is less than \"%s\".\n", str1, str2);
   printf("\"%s\" is greater than \"%s\".\n", str1, str2);
 return 0;
/***** The output should be similar to: *******
Strings compared equal.
*****************************
```

Related Information

• "strcasecmp() — Compare Strings without Case Sensitivity" on page 323

- "strncmp() Compare Strings" on page 344
- "stricmp Compare Strings without Case Sensitivity" on page 340
- "wcscmp() Compare Wide-Character Strings" on page 403
- "wcsncmp() Compare Wide-Character Strings" on page 411
- "_wcsicmp Compare Wide Character Strings without Case Sensitivity" on page 413
- "_wcsnicmp Compare Wide Character Strings without Case Sensitivity" on page 415
- "<strings.h>" on page 17

strncat() — Concatenate Strings

Format

```
#include <string.h>
char *strncat(char *string1, const char *string2, size t count);
```

Language Level: ANSI

Description

The strncat() function appends the first *count* characters of *string2* to *string1* and ends the resulting string with a null character (\0). If *count* is greater than the length of *string2*, the length of *string2* is used in place of *count*.

The strncat() function operates on null-ended strings. The string argument to the function should contain a null character (\0) marking the end of the string.

Return Value

The strncat() function returns a pointer to the joined string (*string1*).

Example that uses strncat()

This example demonstrates the difference between strcat() and strncat(). The strcat() function appends the entire second string to the first, whereas strncat() appends only the specified number of characters in the second string to the first.

```
#include <stdio.h>
#include <string.h>
#define SIZE 40
int main(void)
 char buffer1[SIZE] = "computer";
 char * ptr;
 /* Call strcat with buffer1 and "program" */
 ptr = strcat( buffer1, " program" );
 printf( "strcat : buffer1 = \"%s\"\n", buffer1 );
 /* Reset buffer1 to contain just the string "computer" again */
 memset( buffer1, '\0', sizeof( buffer1 ));
 ptr = strcpy( buffer1, "computer" );
 /* Call strncat with buffer1 and " program" */ ptr = strncat( buffer1, " program", 3 );
 printf( "strncat: buffer1 = \"%s\"\n", buffer1 );
/****** Output should be similar to: *********
strcat : buffer1 = "computer program"
strncat: buffer1 = "computer pr"
```

Related Information

- "strcat() Concatenate Strings" on page 324
- "strncmp() Compare Strings"
- "strncpy() Copy Strings" on page 346
- "strpbrk() Find Characters in String" on page 348
- "strrchr() Locate Last Occurrence of Character in String" on page 354
- "strspn() —Find Offset of First Non-matching Character" on page 355
- "wcscat() Concatenate Wide-Character Strings" on page 400
- "wcsncat() Concatenate Wide-Character Strings" on page 410
- "<string.h>" on page 17

strncmp() — Compare Strings

Format

```
#include <string.h>
int strncmp(const char *string1, const char *string2, size t count);
```

Language Level: ANSI

Description

The strncmp() function compares *string1* and *string2* to the maximum of *count*.

Return Value

The strncmp() function returns a value indicating the relationship between the strings, as follows:

Value	Meaning
Less than 0	string1 less than string2
0	string1 equivalent to string2
Greater than 0	string1 greater than string2

Example that uses strncmp()

This example demonstrates the difference between the strcmp() function and the strncmp() function.

```
#include <stdio.h>
#include <string.h>
#define SIZE 10
int main(void)
 int result;
 int index = 3;
 char buffer1[SIZE] = "abcdefg";
 char buffer2[SIZE] = "abcfg";
 void print result( int, char *, char * );
 result = strcmp( buffer1, buffer2 );
 printf( "Comparison of each character\n" );
 printf( " strcmp: " );
 print_result( result, buffer1, buffer2 );
 result = strncmp( buffer1, buffer2, index);
 printf( "\nComparison of only the first %i characters\n", index );
 printf( " strncmp: " );
 print result( result, buffer1, buffer2 );
void print result( int res, char * p buffer1, char * p buffer2 )
 if ( res == 0 )
   printf( "\"%s\" is identical to \"%s\"\n", p buffer1, p buffer2);
 else if ( res < 0 )
   printf( "\"%s\" is less than \"%s\"\n", p buffer1, p buffer2 );
 else
   printf( "\"%s\" is greater than \"%s\"\n", p buffer1, p buffer2 );
/****** Output should be similar to: *********
Comparison of each character
 strcmp: "abcdefg" is less than "abcfg"
Comparison of only the first 3 characters
 strncmp: "abcdefg" is identical to "abcfg"
```

Related Information

- "strcmp() Compare Strings" on page 326
- "strcspn() Find Offset of First Character Match" on page 331
- "strncat() Concatenate Strings" on page 343
- "strncpy() Copy Strings" on page 346
- "strpbrk() Find Characters in String" on page 348

- "strrchr() Locate Last Occurrence of Character in String" on page 354
- "strspn() —Find Offset of First Non-matching Character" on page 355
- "wcscmp() Compare Wide-Character Strings" on page 403
- "wcsncmp() Compare Wide-Character Strings" on page 411
- "<string.h>" on page 17
- "_wcsicmp Compare Wide Character Strings without Case Sensitivity" on page 413
- "_wcsnicmp Compare Wide Character Strings without Case Sensitivity" on page 415

strncpy() — Copy Strings

Format

```
#include <string.h>
char *strncpy(char *string1, const char *string2, size t count);
```

Language Level: ANSI

Description

The strncpy() function copies count characters of string2 to string1. If count is less than or equal to the length of string2, a null character (\0) is not appended to the copied string. If count is greater than the length of string2, the string1 result is padded with null characters (\0) up to length *count*.

Return Value

The strncpy() function returns a pointer to *string1*.

Example that uses strncpy()

This example demonstrates the difference between strcpy() and strncpy().

```
#include <stdio.h>
#include <string.h>
#define SIZE
int main(void)
  char source[ SIZE ] = "123456789";
  char source1[ SIZE ] = "123456789";
  char destination[ SIZE ] = "abcdefg";
  char destination1[ SIZE ] = "abcdefg";
  char * return string;
       index = 5;
  int
  /* This is how strcpy works */
  printf( "destination is originally = '%s'\n", destination );
  return_string = strcpy( destination, source );
  printf( "After strcpy, destination becomes 'ss'\n\n", destination );
  /* This is how strncpy works */
  printf( "destination1 is originally = \frac{1}{8}s'\n", destination1 );
  return_string = strncpy( destination1, source1, index );
  printf( "After strncpy, destination1 becomes '%s'\n", destination1 );
/****** Output should be similar to: *********
destination is originally = 'abcdefg'
After strcpy, destination becomes '123456789'
destination1 is originally = 'abcdefg'
After strncpy, destination1 becomes '12345fg'
```

- "strcpy() Copy Strings" on page 330
- "strcspn() Find Offset of First Character Match" on page 331
- "strncat() Concatenate Strings" on page 343
- "strncmp() Compare Strings" on page 344
- "strpbrk() Find Characters in String" on page 348
- "strrchr() Locate Last Occurrence of Character in String" on page 354
- "strspn() —Find Offset of First Non-matching Character" on page 355
- "<string.h>" on page 17

strnicmp - Compare Substrings Without Case Sensitivity

Format

```
#include <string.h>
int strnicmp(const char *string1, const char *string2, int n);
```

Note: The strnicmp function is supported only for C++, not for C.

Language Level: Extension

Description

strnicmp compares, at most, the first *n* characters of *string1* and *string2* without sensitivity to case.

The function operates on null terminated strings. The string arguments to the function are expected to contain a null character (\0) marking the end of the string.

Return Value

strnicmp returns a value indicating the relationship between the substrings, as follows:

*/

Value Meaning Less than 0 substring1 less than substring2 substring1 equivalent to substring2 Greater than 0 substring1 greater than substring2

Example that uses strnicmp()

This example uses strnicmp to compare two strings.

```
#include <stdio.h>
#include <string.h>
int main(void)
   char *str1 = "THIS IS THE FIRST STRING";
char *str2 = "This is the second string";
   int numresult;
     /* Compare the first 11 characters of str1 and str2
        without regard to case
   numresult = strnicmp(str1, str2, 11);
   if (numresult < 0)
      printf("String 1 is less than string2.\n");
   else
      if (numresult > 0)
         printf("String 1 is greater than string2.\n");
         printf("The two strings are equivalent.\n");
   return 0;
```

The output should be:

The two strings are equivalent.

Related Information:

- strcmp
- "strcmpi Compare Strings Without Case Sensitivity" on page 328
- "stricmp Compare Strings without Case Sensitivity" on page 340
- strncmp
- wcscmp
- wcsncmp
- <string.h>

strpbrk() — Find Characters in String

Format

```
#include <string.h>
char *strpbrk(const char *string1, const char *string2);
```

Language Level: ANSI

Description

The strpbrk() function locates the first occurrence in the string pointed to by string1 of any character from the string pointed to by string2.

Return Value

The strpbrk() function returns a pointer to the character. If string1 and string2 have no characters in common, a NULL pointer is returned.

Example that uses strpbrk()

This example returns a pointer to the first occurrence in the array string of either a or b.

```
#include <stdio.h>
#include <string.h>
int main(void)
  char *result, *string = "A Blue Danube";
  char *chars = "ab";
  result = strpbrk(string, chars);
  printf("The first occurrence of any of the characters \"%s\" in "
         "\"%s\" is \"%s\"\n", chars, string, result);
/****** Output should be similar to: *********
The first occurrence of any of the characters "ab" in "The Blue Danube"
is "anube"
*/
```

Related Information

- "strchr() Search for Character" on page 325
- "strcmp() Compare Strings" on page 326
- "strcspn() Find Offset of First Character Match" on page 331
- "strncmp() Compare Strings" on page 344
- "strrchr() Locate Last Occurrence of Character in String" on page 354
- "strspn() —Find Offset of First Non-matching Character" on page 355
- "wcschr() Search for Wide Character" on page 402
- "wcscspn() Find Offset of First Wide-Character Match" on page 406
- "wcspbrk() Locate Wide Characters in String" on page 416
- "wcsrchr() Locate Last Occurrence of Wide Character in String" on page 417
- "wcswcs() Locate Wide-Character Substring" on page 432
- "<string.h>" on page 17

strnset - strset - Set Characters in String

Format

```
#include <string.h>
char *strnset(char *string, int c, size_t n);
char *strset(char *string, int c);
```

Note: The strnset and strset functions are supported only for C++, not for C.

Language Level: Extension

Description

strnset sets, at most, the first *n* characters of *string* to *c* (converted to a char). If *n* is greater than the length of string, the length of string is used in place of n. strset sets all characters of *string*, except the ending null character ($\backslash 0$), to c (converted to a char). For both functions, the string is a null-terminated string.

Return Value

Both strset and strnset return a pointer to the altered string. There is no error return value.

Example that uses strnset() and strset()

In this example, strnset sets not more than four characters of a string to the character 'x'. Then the strset function changes any non-null characters of the string to the character 'k'.

```
#include <stdio.h>
#include <string.h>
int main(void)
    char str[] = "abcdefghi";
   printf("This is the string: %s\n", str);
   printf("This is the string after strnset: %s\n", strnset((char*)str, 'x', 4));
printf("This is the string after strset: %s\n", strset((char*)str, 'k'));
    return 0;
```

The output should be:

```
This is the string: abcdefghi
This is the string after strnset: xxxxefghi
This is the string after strset: kkkkkkkkk
```

Related Information:

- strchr
- strpbrk
- wcschr
- wcspbrk
- <string.h>

strptime()— Convert String to Date/Time

Format

```
#include <time.h>
char *strptime(const char *buf, const char *format, struct tm *tm);
```

Language Level: XPG4

Thread Safe: YES.

Description

The strptime() function converts the character string pointed to by *buf* to values that are stored in the *tm* structure pointed to by *tm*, using the format specified by *format*.

1

ı

I

The *format* contains zero or more directives. A directive contains either an ordinary character (not % or a white space), or a conversion specification. Each conversion specification is composed of a % character followed by one or more conversion characters, which specify the replacement required. There must be a white space or other delimiter in both *buf* and *format* to be guaranteed that the function will behave as expected. There must be a delimiter between two string-to-number conversions, or the first number conversion may convert characters that belong to the second conversion specifier.

Any whitespace (as specified by isspace()) encountered before a directive is scanned in either the format string or the input string will be ignored. A directive that is an ordinary character must exactly match the next scanned character in the input string. Case is relevant when matching ordinary character directives. If the ordinary character directive in the format string does not match the character in the input string, strptime is not successful. No more characters will be scanned.

Any other conversion specification is matched by scanning characters in the input string until a character that is not a possible character for that specification is found or until no more characters can be scanned. If the specification was string-to-number, the possible character range is +,- or a character specified by isdigit(). Number specifiers do not require leading zeros. If the specification needs to match a field in the current locale, scanning is repeated until a match is found. Case is ignored when matching fields in the locale. If a match is found, the structure pointed to by *tm* will be updated with the corresponding locale information. If no match is found, strptime is not successful. No more characters will be scanned.

Missing fields in the *tm* structure may be filled in by strftime if given enough information. For example, if a date is given, tm yday can be calculated.

Each standard conversion specification is replaced by appropriate characters as described in the following table:

Specifier	Meaning
%a	Name of day of the week, can be either the full name or an abbreviation.
%A	Same as %a.
%b	Month name, can be either the full name or an abbreviation.
%B	Same as %b.
%с	Date/time, in the format of the locale.
%C	Century number [00–99]. Calculates the year if a two-digit year is used.
%d	Day of the month [1–31].
%D	Date format, same as %m/%d/%y.
%e	Same as %d.
%h	Same as %b.
%Н	Hour in 24-hour format [0–23].
%I	Hour in 12-hour format [1-12].
%j	Day of the year [1-366].

I
I
I
I
l

I
I
l
I
I
'
I
I
I
I
I
I

Specifier	Meaning
%m	Month [1-12].
%M	Minute [0-59].
%n	Skip all whitespaces until a newline character is found.
%p	AM or PM string, used for calculating the hour if 12-hour format is used.
%r	Time in AM/PM format of the locale. If not available in the locale time format, defaults to the POSIX time AM/PM format: %I:%M:%S %p.
%R	24-hour time format without seconds, same as %H:%M.
%S	Second [00-61]. The range for seconds allows for a leap second and a double leap second.
%t	Skip all whitespaces until a tab character is found.
%T	24 hour time format with seconds, same as %H:%M:%S.
%u	Weekday [1–7]. Monday is 1 and Sunday is 7.
%U	Week number of the year [1-53], Sunday is the first day of the week. Used in calculating the day of the year.
%V	ISO week number of the year [1-53]. Monday is the first day of the week. If the week containing January 1st has four or more days in the new year, it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1 of the new year. Used in calculating the day of the year.
%w	Weekday [0 -6]. Sunday is 0.
%W	Week number of the year [0-53]. Monday is the first day of the week. Used in calculating the day of the year.
%x	Date in the format of the locale.
%X	Time in the format of the locale.
%y	2-digit year [0-99].
%Y	4-digit year. Can be negative.
%Z	Time zone name. Available only from the locale. Used to determine if the locale is in daylight savings time.
%%	% character.

Modified Conversion Specifiers

Some conversion specifiers can be modified by the E or O modifier characters to indicate that an alternate format or specification should be used. If a modified conversion specifier uses a field in the current locale that is unavailable, then the behavior will be as if the unmodified conversion specification were used. For example, if the era string is the empty string "", which means that era is unavailable, then %EY would act like %Y.

Specifier	Meaning
%Ec	Date/time for current era.
%EC	Era name.
%Ex	Date for current era.
%EX	Time for current era.
%Ey	Era year. This is the offset from the base year.

Specifier	Meaning
%EY	Year for the current era.
%Od	Day of the month using alternate digits.
%Oe	Same as %Od.
%OH	Hour in 24-hour format using alternate digits.
%OI	Hour in 12-hour format using alternate digits.
%Om	Month using alternate digits.
%OM	Minutes using alternate digits.
%OS	Seconds using alternate digits.
%Ou	Day of the week using alternate digits. Monday is 1 and Sunday is 7.
%OU	Week number of the year using alternate digits. Sunday is the first day of the week.
%Ow	Weekday using alternate digit. Sunday is 0 and Saturday is 6.
%OW	Week number of the year using alternate digits. Monday is the first day of the week.
%Oy	2-digit year using alternate digits.

Note: This function is only accessible if LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

Return Value

| | |

On successful completion, the strptime() function returns a pointer to the character following the last character parsed. Otherwise, a null pointer is returned.

Example that uses strptime()

```
#include <stdio.h>
#include <locale.h>
#include <time.h>
int main(void)
    char buf[100];
    time t t;
    struct tm *timeptr,result;
    setlocale(LC_ALL,"/QSYS.LIB/EN_US.LOCALE");
    t = time(NULL);
    timeptr = localtime(&t);
    strftime(buf,sizeof(buf), "%a %m/%d/%Y %r", timeptr);
    if(strptime(buf, "%a %m/%d/%Y %r",&result) == NULL)
           printf("\nstrptime failed\n");
   else
           printf("tm_hour: %d\n",result.tm_hour);
           printf("tm min: %d\n",result.tm min);
           printf("tm sec: %d\n",result.tm sec);
           printf("tm_mon: %d\n",result.tm_mon);
           printf("tm_mday: %d\n",result.tm_mday);
           printf("tm_year: %d\n",result.tm_year);
printf("tm_yday: %d\n",result.tm_yday);
printf("tm_wday: %d\n",result.tm_wday);
   }
```

```
return 0:
/***********************
    The output should be similar to:
    Tue 10/30/2001 10:59:10 AM
    tm hour: 10
    tm min: 59
    tm_sec: 10
    tm mon: 9
    tm_mday: 30
tm_year: 101
    tm_yday: 302
    tm wday: 2
```

- "asctime() Convert Time to Character String" on page 40
- "asctime_r() Convert Time to Character String (Restartable)" on page 42
- "ctime() Convert Time to Character String" on page 66
- "ctime_r() Convert Time to Character String (Restartable)" on page 67
- "gmtime() Convert Time" on page 141
- "gmtime_r() Convert Time (Restartable)" on page 143
- "localtime() Convert Time" on page 163
- "localtime_r() Convert Time (Restartable)" on page 164
- "setlocale() Set Locale" on page 308
- "strftime() Convert Date/Time to String" on page 336
- "time() Determine Current Time" on page 373
- "<time.h>" on page 17

strrchr() — Locate Last Occurrence of Character in String

Format

```
#include <string.h>
char *strrchr(const char *string, int c);
```

Language Level: ANSI

Description

The strrchr() function finds the last occurrence of *c* (converted to a character) in string. The ending null character is considered part of the string.

Return Value

The strrchr() function returns a pointer to the last occurrence of *c* in *string*. If the given character is not found, a NULL pointer is returned.

Example that uses strrchr()

This example compares the use of strchr() and strrchr(). It searches the string for the first and last occurrence of p in the string.

```
#include <stdio.h>
#include <string.h>
#define SIZE 40
int main(void)
 char buf[SIZE] = "computer program";
 char * ptr;
 int ch = 'p';
 /* This illustrates strchr */
 ptr = strchr( buf, ch );
 printf( "The first occurrence of %c in '%s' is '%s'\n", ch, buf, ptr );
 /* This illustrates strrchr */
 ptr = strrchr( buf, ch );
 printf( "The last occurrence of %c in '%s' is '%s'\n", ch, buf, ptr );
/******* Output should be similar to: **********
The first occurrence of p in 'computer program' is 'puter program'
The last occurrence of p in 'computer program' is 'program'
```

- "strchr() Search for Character" on page 325
- "strcmp() Compare Strings" on page 326
- "strcspn() Find Offset of First Character Match" on page 331
- "strncmp() Compare Strings" on page 344
- "strpbrk() Find Characters in String" on page 348
- "strspn() —Find Offset of First Non-matching Character"
- "<string.h>" on page 17
- wcsrchar()

strspn() —Find Offset of First Non-matching Character

Format

```
#include <string.h>
size_t strspn(const char *string1, const char *string2);
```

Language Level: ANSI

Thread Safe: YES

Description

The strspn() function finds the first occurrence of a character in *string1* that is not contained in the set of characters that is specified by *string2*. The null character (\0) that ends *string2* is not considered in the matching process.

Return Value

The strspn() function returns the index of the first character found. This value is equal to the length of the initial substring of *string1* that consists entirely of

characters from string2. If string1 begins with a character not in string2, the strspn() function returns 0. If all the characters in *string1* are found in *string2*, the length of *string1* is returned.

Example that uses strspn()

This example finds the first occurrence in the array string of a character that is not an a, b, or c. Because the string in this example is cabbage, the strspn() function returns 5, the length of the segment of cabbage before a character that is not an a, b, or c.

```
#include <stdio.h>
#include <string.h>
int main(void)
 char * string = "cabbage";
 char * source = "abc";
 int index;
 index = strspn( string, "abc" );
 printf( "The first %d characters of \"%s\" are found in \"%s\"\n",
         index, string, source );
/****** Output should be similar to: *********
The first 5 characters of "cabbage" are found in "abc"
```

Related Information

- "strcat() Concatenate Strings" on page 324
- "strchr() Search for Character" on page 325
- "strcmp() Compare Strings" on page 326
- "strcpy() Copy Strings" on page 330
- "strcspn() Find Offset of First Character Match" on page 331
- "strpbrk() Find Characters in String" on page 348
- "strrchr() Locate Last Occurrence of Character in String" on page 354
- "wcschr() Search for Wide Character" on page 402
- "wcscspn() Find Offset of First Wide-Character Match" on page 406
- "wcspbrk() Locate Wide Characters in String" on page 416
- "wcsspn() Find Offset of First Non-matching Wide Character" on page 420
- "wcswcs() Locate Wide-Character Substring" on page 432
- "wcsrchr() Locate Last Occurrence of Wide Character in String" on page 417
- "<string.h>" on page 17

strstr() — Locate Substring

Format

```
#include <string.h>
char *strstr(const char *string1, const char *string2);
```

Language Level: ANSI

Thread Safe: YES

Description

The strstr() function finds the first occurrence of *string2* in *string1*. The function ignores the null character (\0) that ends *string2* in the matching process.

Return Value

The strstr() function returns a pointer to the beginning of the first occurrence of *string2* in *string1*. If *string2* does not appear in *string1*, the strstr() function returns NULL. If *string2* points to a string with zero length, the strstr() function returns *string1*.

Example that uses strstr()

This example locates the string "haystack" in the string "needle in a haystack".

Related Information

- "strchr() Search for Character" on page 325
- "strcmp() Compare Strings" on page 326
- "strcspn() Find Offset of First Character Match" on page 331
- "strncmp() Compare Strings" on page 344
- "strpbrk() Find Characters in String" on page 348
- "strrchr() Locate Last Occurrence of Character in String" on page 354
- "strspn() —Find Offset of First Non-matching Character" on page 355
- "wcschr() Search for Wide Character" on page 402
- "wcscspn() Find Offset of First Wide-Character Match" on page 406
- "wcspbrk() Locate Wide Characters in String" on page 416
- "wcsrchr() Locate Last Occurrence of Wide Character in String" on page 417
- "wcsspn() Find Offset of First Non-matching Wide Character" on page 420
- "wcswcs() Locate Wide-Character Substring" on page 432
- "<string.h>" on page 17

strtod() — Convert Character String to Double

Format

```
#include <stdlib.h>
double strtod(const char *nptr, char **endptr);
```

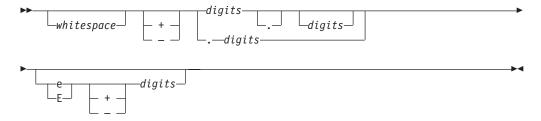
Language Level: ANSI

Thread Safe: YES

Description

The strtod() function converts a character string to a double-precision value. The parameter nptr points to a sequence of characters that can be interpreted as a numeric value of the type double. This function stops reading the string at the first character that it cannot recognize as part of a number. This character can be the null character at the end of the string.

The strtod() function expects *nptr* to point to a string with the following form:



The first character that does not fit this form stops the scan.

The behavior of this function is affected by the LC_NUMERIC category of the current locale.

Return Value

The strtod() function returns the value of the floating-point number, except when the representation causes an underflow or overflow. For an overflow, it returns -HUGE_VAL or +HUGE_VAL; for an underflow, it returns 0.

In both cases, errno is set to ERANGE, depending on the base of the value. If the string pointed to by *nptr* does not have the expected form, no conversion is performed and the value of *nptr* is stored in the object pointed to by *endptr*, provided that *endptr* is not a NULL pointer.

The strtod() function does not fail if a character other than a digit follows an E or e read in as an exponent. For example, 100elf will be converted to the floating-point value 100.0.

Example that uses strtod()

This example converts the strings to a double value. It prints out the converted value and the substring that stopped the conversion.

```
#include <stdlib.h>
#include <stdio.h>
int main(void)
  char *string, *stopstring;
  double x;
  string = "3.1415926This stopped it";
  x = strtod(string, &stopstring);
  printf("string = %s\n", string);
  printf(" strtod = %lf\n", x);
  printf("
            Stopped scan at %s\n\n", stopstring);
  string = "100ergs";
  x = strtod(string, &stopstring);
  printf("string = \"%s\"\n", string);
  printf(" strtod = %lf\n", x);
  printf("
            Stopped scan at \"%s\"\n\n", stopstring);
/****** Output should be similar to: *********
string = 3.1415926This stopped it
  strtod = 3.141593
  Stopped scan at This stopped it
string = 100ergs
  strtod = 100.000000
  Stopped scan at ergs
```

- "atof() Convert Character String to Float" on page 47
- "atoi() Convert Character String to Integer" on page 49
- "atol() atoll() Convert Character String to Long or Long Long Integer" on page 50
- "strtol() strtoll() Convert Character String to Long and Long Long Integer" on page 362
- "strtoul() strtoull() Convert Character String to Unsigned Long and Unsigned Long Integer" on page 364
- "<stdlib.h>" on page 16
- "wcstod() Convert Wide-Character String to Double" on page 422

strtok() — Tokenize String

Format

```
#include <string.h>
char *strtok(char *string1, const char *string2);
```

Language Level: ANSI

Description

Thread Safe: NO. Use strtok_r() instead.

The strtok() function reads *string1* as a series of zero or more tokens, and *string2* as the set of characters serving as delimiters of the tokens in *string1*. The tokens in

string1 can be separated by one or more of the delimiters from string2. The tokens in *string1* can be located by a series of calls to the strtok() function.

In the first call to the strtok() function for a given *string1*, the strtok() function searches for the first token in string1, skipping over leading delimiters. A pointer to the first token is returned.

When the strtok() function is called with a NULL string1 argument, the next token is read from a stored copy of the last non-null *string1* parameter. Each delimiter is replaced by a null character. The set of delimiters can vary from call to call, so *string2* can take any value. Note that the initial value of *string1* is not preserved after the call to the strtok() function.

Note that the strtok() function writes data into the buffer. The function should be passed to a non-critical buffer containing the string to be tokenized because the buffer will be damaged by the strtok() function.

Return Value

The first time the strtok() function is called, it returns a pointer to the first token in string1. In later calls with the same token string, the strtok() function returns a pointer to the next token in the string. A NULL pointer is returned when there are no more tokens. All tokens are null-ended.

Note: The strtok() function uses an internal static pointer to point to the next token in the string being tokenized. A reentrant version of the strtok() function, strtok r(), which does not use any internal static storage, can be used in place of the strtok() function.

Example that uses strtok()

Using a loop, this example gathers tokens, separated by commas, from a string until no tokens are left. The example prints the tokens, a string, of, and tokens.

```
#include <stdio.h>
#include <string.h>
int main(void)
  char *token, *string = "a string, of, ,tokens\0,after null terminator";
  /* the string pointed to by string is broken up into the tokens
      "a string", " of", " ", and "tokens" ; the null terminator (\setminus 0)
     is encountered and execution stops after the token "tokens"
  token = strtok(string, ",");
     printf("token: %s\n", token);
  while (token = strtok(NULL, ","));
/****** Output should be similar to: *********
token: a string
token: of
token:
token: tokens
*/
```

- "strcat() Concatenate Strings" on page 324
- "strchr() Search for Character" on page 325
- "strcmp() Compare Strings" on page 326
- "strcpy() Copy Strings" on page 330
- "strcspn() Find Offset of First Character Match" on page 331
- "strspn() —Find Offset of First Non-matching Character" on page 355
- "strtok_r() Tokenize String (Restartable)"
- "<string.h>" on page 17

strtok_r() — Tokenize String (Restartable)

Format

ı

Language Level: XPG4

Thread Safe: YES.

Description

This function is the restartable version of strtok().

The strtok_r() function reads *string* as a series of zero or more tokens, and *seps* as the set of characters serving as delimiters of the tokens in *string*. The tokens in *string* can be separated by one or more of the delimiters from *seps*. The arguments *lasts* points to a user-provided pointer, which points to stored information necessary for the strtok_r() function to continue scanning the same string.

In the first call to the strtok_r() function for a given null-ended *string*, it searches for the first token in *string*, skipping over leading delimiters. It returns a pointer to the first character of the first token, writes a null character into *string* immediately following the returned token, and updates the pointer to which *lasts* points.

To read the next token from *string*, call the <code>strtok_r()</code> function with a NULL *string* argument. This causes the <code>strtok_r()</code> function to search for the next token in the previous token string. Each delimiter is replaced in the original *string* is replaced by a null character, and the pointer to which *lasts* points is updated. The set of delimiters in *seps* can vary from call to call, but *lasts* must remain unchanged from the previous call. When no tokens remain in *string*, a NULL pointer is returned.

Return Value

The first time the strtok_r() function is called, it returns a pointer to the first token in *string*. In later calls with the same token string, the strtok_r() function returns a pointer to the next token in the string. A NULL pointer is returned when there are no more tokens. All tokens are null-ended.

Related Information

- "strcat() Concatenate Strings" on page 324
- "strchr() Search for Character" on page 325

- "strcmp() Compare Strings" on page 326
- "strcpy() Copy Strings" on page 330
- "strcspn() Find Offset of First Character Match" on page 331
- "strspn() —Find Offset of First Non-matching Character" on page 355
- "strtok() Tokenize String" on page 359
- "<string.h>" on page 17

strtol() — strtoll() — Convert Character String to Long and Long Long Integer

```
Format (strtol())
#include <stdlib.h>
long int strtol(const char *nptr, char **endptr, int base);
Format (strtoll())
#include <stdlib.h>
long long int strtoll(char *string, char **endptr, int base);
Language Level: ANSI
```

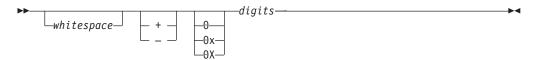
Thread Safe: YES

Description

The strtol() function converts a character string to a long integer value. The parameter *nptr* points to a sequence of characters that can be interpreted as a numeric value of type long int.

The strtoll() function converts a character string to a long long integer value. The parameter *nptr* points to a sequence of characters that can be interpreted as a numeric value of type long long int.

When you use these functions, the *nptr* parameter should point to a string with the following form:



If the base parameter is a value between 2 and 36, the subject sequence's expected form is a sequence of letters and digits representing an integer whose radix is specified by the base parameter. This sequence is optionally preceded by a positive (+) or negative (-) sign. Letters from a to z inclusive (either upper or lower case) are ascribed the values 10 to 35; only letters whose ascribed values are less than that of the base parameter are permitted. If the base parameter has a value of 16, the characters 0x or 0X optionally precede the sequence of letters and digits, following the positive (+) or negative (-) sign, if present.

If the value of the base parameter is 0, the string determines the base. After an optional leading sign a leading 0 indicates octal conversion, a leading 0x or 0X indicates hexadecimal conversion, and all other leading characters result in decimal conversion.

These functions scan the string up to the first character that is inconsistent with the *base* parameter. This character may be the null character (' $\0$ ') at the end of the string. Leading white-space characters are ignored, and an optional sign may precede the digits.

If the value of the *endptr* parameter is not null a pointer, a pointer to the character that ended the scan is stored in the value pointed to by *endptr*. If a value cannot be formed, the value pointed to by *endptr* is set to the *nptr* parameter

Return Value

If base has an invalid value (less than 0, 1, or greater than 36), errno is set to EDOM and 0 is returned. The value pointed to by the *endptr* parameter is set to the value of the *nptr* parameter.

If the value is outside the range of representable values, errno is set to ERANGE. If the value is positive, the strtol() function will return LONG_MAX, and the strtoll() function will return LONGLONG_MAX. If the value is negative, the strtol() function will return LONG_MIN, and the strtoll() function will return LONGLONG_MIN.

If no characters are converted, the strtoll() function will set errno to EINVAL and 0 is returned. The strtol() function will return 0 but will not set errno to EINVAL. In both cases the value pointed to by *endptr* is set to the value of the *nptr* parameter. Upon successful completion, both functions return the converted value.

Example that uses strtol()

This example converts the strings to a long value. It prints out the converted value and the substring that stopped the conversion.

```
#include <stdlib.h>
#include <stdio.h>
int main(void)
  char *string, *stopstring;
  long 1;
  int bs;
  string = "10110134932";
  printf("string = %s\n", string);
  for (bs = 2; bs <= 8; bs *= 2)
     1 = strtol(string, &stopstring, bs);
     printf(" strtol = %1d (base %d)\n", 1, bs);
     printf(" Stopped scan at %s\n\n", stopstring);
/******* Output should be similar to: *********
string = 10110134932
  strtol = 45 (base 2)
  Stopped scan at 34932
  strtol = 4423 (base 4)
  Stopped scan at 4932
```

Related Information

- "atof() Convert Character String to Float" on page 47
- "atoi() Convert Character String to Integer" on page 49
- "atol() atoll() Convert Character String to Long or Long Long Integer" on page 50
- "strtod() Convert Character String to Double" on page 357
- "strtoul() strtoull() Convert Character String to Unsigned Long and Unsigned Long Long Integer"
- "<stdlib.h>" on page 16
- "wcstol() wcstoll() Convert Wide Character String to Long and Long Long Integer" on page 424

strtoul() — strtoull() — Convert Character String to Unsigned Long and **Unsigned Long Long Integer**

```
Format (strtoul())
```

#include <stdlib.h> unsigned long int strtoul(const char *nptr, char **endptr, int base);

Format (strtoull())

#include <stdlib.h> unsigned long long int strtoull(char *string, char **endptr, int base);

Language Level: ANSI

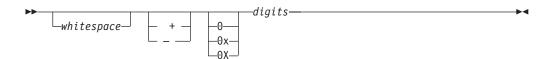
Thread Safe: YES

Description

The strtoul () function converts a character string to an unsigned long integer value. The parameter *nptr* points to a sequence of characters that can be interpreted as a numeric value of type unsigned long int.

The strtoull() function converts a character string to an unsigned long long integer value. The parameter nptr points to a sequence of characters that can be interpreted as a numeric value of type unsigned long long int.

When you use these functions, the *nptr* parameter should point to a string with the following form:



If the base parameter is a value between 2 and 36, the subject sequence's expected form is a sequence of letters and digits representing an integer whose radix is specified by the base parameter. This sequence is optionally preceded by a positive (+) or negative (-) sign. Letters from a to z inclusive (either upper or lower case) are ascribed the values 10 to 35. Only letters whose ascribed values are less than that of the base parameter are permitted. If the base parameter has a value of 16 the characters 0x or 0X optionally precede the sequence of letters and digits, following the positive (+) or negative (-) sign, if present.

If the value of the *base* parameter is 0, the string determines the *base*. After an optional leading sign a leading 0 indicates octal conversion, a leading 0x or 0x indicates hexadecimal conversion, and all other leading characters result in decimal conversion.

These functions scan the string up to the first character that is inconsistent with the base parameter. This character may be the null character ('\0') at the end of the string. Leading white-space characters are ignored, and an optional sign may precede the digits.

If the value of the *endptr* parameter is not null a pointer, a pointer to the character that ended the scan is stored in the value pointed to by endptr. If a value cannot be formed, the value pointed to by *endptr* is set to the *nptr* parameter.

Return Value

1

I

I

ı

1

I

ı

1

If *base* has an invalid value (less than 0, 1, or greater than 36), errno is set to EDOM and 0 is returned. The value pointed to by the *endptr* parameter is set to the value of the *nptr* parameter.

If the value is outside the range of representable values, errno is set to ERANGE. The strtoul() function will return ULONG_MAX and the strtoull() function will return ULONGLONG_MAX.

If no characters are converted, the strtoull() function will set errno to EINVAL and 0 is returned. The strtoul() function will return 0 but will not set errno to EINVAL. In both cases the value pointed to by *endptr* is set to the value of the *nptr* parameter. Upon successful completion, both functions return the converted value.

Example that uses strtoul()

This example converts the string to an unsigned long value. It prints out the converted value and the substring that stopped the conversion.

```
#include <stdio.h>
#include <stdlib.h>
#define BASE 2
int main(void)
  char *string, *stopstring;
  unsigned long ul;
  string = "1000e13 e";
  printf("string = %s\n", string);
  ul = strtoul(string, &stopstring, BASE);
  printf(" strtoul = %ld (base %d)\n", ul, BASE);
  printf("
           Stopped scan at %s\n\n", stopstring);
/******* Output should be similar to: *********
string = 1000e13 e
  strtoul = 8 (base 2)
  Stopped scan at e13 e
```

Related Information

"atof() — Convert Character String to Float" on page 47

- "atoi() Convert Character String to Integer" on page 49
- "atol() atoll() Convert Character String to Long or Long Long Integer" on page 50
- "strtod() Convert Character String to Double" on page 357
- "strtol() strtoll() Convert Character String to Long and Long Long Integer" on page 362
- "<stdlib.h>" on page 16
- "wcstoul() wcstoull() Convert WideCharacter String to Unsigned Long and Unsigned Long Long Integer" on page 430

strxfrm() — Transform String

Format

```
#include <string.h>
size t strxfrm(char *string1, const char *string2, size t count);
```

Language Level: ANSI

Thread Safe: YES

Description

The strxfrm() function transforms the string pointed to by string2 and places the result into the string pointed to by string1. The transformation is determined by the program's current locale. The transformed string is not necessarily readable, but can be used with the strcmp() or the strncmp() functions.

The behavior of this function is affected by the LC_COLLATE category of the current locale.

Return Value

The strxfrm() function returns the length of the transformed string, excluding the ending null character. If the returned value is greater than or equal to count, the contents of the transformed string are indeterminate.

If strxfrm() is unsuccessful, errno is changed. The value of errno may be set to EINVAL (the string1 or string2 arguments contain characters which are not available in the current locale).

Example that uses strxfrm()

This example prompts the user to enter a string of characters, then uses strxfrm()to transform the string and return its length.

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char *string1, buffer[80];
    int length;

    printf("Type in a string of characters.\n ");
    string1 = gets(buffer);
    length = strxfrm(NULL, string1, 0);
    printf("You would need a %d element array to hold the string\n",length);
    printf("\n\n%s\n\n transformed according",string1);
    printf(" to this program's locale. \n");
}
```

- "localeconv() Retrieve Information from the Environment" on page 158
- "setlocale() Set Locale" on page 308
- "strcmp() Compare Strings" on page 326
- "strcoll() Compare Strings" on page 329
- "strncmp() Compare Strings" on page 344
- "<string.h>" on page 17

swprintf() — Format and Write Wide Characters to Buffer

Format

Language Level: ANSI

Thread Safe: YES

Description

I

1

The swprintf() function formats and stores a series of wide characters and values into the wide-character buffer *wcsbuffer*. The swprintf() function is equivalent to the sprintf() function, except that it operates on wide characters.

The value n specifies the maximum number of wide characters to be written, including the ending null character. The swprintf() function converts each entry in the argument-list according to the corresponding wide-character format specifier in format. The format has the same form and function as the format string for the printf() function, with the following exceptions:

- %c (without an l prefix) converts a character argument to wchar_t, as if by calling the mbtowc() function.
- %lc and %C copy a wchar_t to wchar_t. %#lc and %#C are equivalent to %lc and %C, respectively.
- %s (without an l prefix) converts an array of multibyte characters to an array of wchar_t, as if by calling the mbstowcs() function. The array is written up to, but not including, the ending null character, unless the precision specifies a shorter output.

367

Width and precision always are wide characters.

When the program is compiled with LOCALETYPE(*LOCALEUCS2) and SYSIFCOPT(*IFSIO), the wide characters that are written by this function are UNICODE characters, and the inputs for %lc and %ls are assumed to be UNICODE characters. To view UNICODE examples, see "fwprintf() — Format Data as Wide Characters and Write to a Stream" on page 123.

A null wide character is added to the end of the wide characters written; the null wide character is not counted as part of the returned value. If copying takes place between objects that overlap, the behavior is undefined.

In extended mode, the swprintf() function also converts floating-point values of NaN and infinity to the strings "NAN" or "nan" and "INFINITY" or "infinity". The case and sign of the string is determined by the format specifiers.

Note: This function is available only when LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) and SYSIFCOPT(*IFSIO) is specified on the compilation command.

Return Value

The swprintf() function returns the number of wide characters that are written in the array, not counting the ending null wide character.

The value of errno may be set to **EINVAL**, invalid argument.

Example that uses swprintf()

This example uses the swprintf() function to format and print several values to buffer.

```
#include <wchar.h>
#include <stdio.h>
#define BUF SIZE 100
int main(void)
  wchar t wcsbuf[BUF SIZE];
  wchar_t wstring[] = L"ABCDE";
  int
          num:
  num = swprintf(wcsbuf, BUF SIZE, L"%s", "xyz");
  num += swprintf(wcsbuf + num, BUF_SIZE - num, L"%1s", wstring);
  num += swprintf(wcsbuf + num, BUF_SIZE - num, L"%i", 100);
  printf("The array wcsbuf contains: \"%1s\"\n", wcsbuf);
  return 0;
  /***********************
     The output should be similar to:
     The array wcsbuf contains: "xyzABCDE100"
```

- "printf() Print Formatted Characters" on page 200
- "sprintf() Print Formatted Data to Buffer" on page 317
- "vswprintf() Format and Write Wide Characters to Buffer" on page 393
- "<wchar.h>" on page 18

swscanf() — Read Wide Character Data

Format

```
#include <wchar.h>
int swscanf(const wchar t *buffer, const wchar t *format, argument-list);
```

Language Level: ANSI

Thread Safe: YES

Description

The swscanf() function is equivalent of the fwscanf() function, except that the argument buffer specifies a wide string from which the input is to be obtained, rather than from a stream. Reaching the end of the wide string is equivalent to encountering end-of-file for the fwscanf() function.

Return Value

The swscanf() function returns the number of fields that were successfully converted and assigned. The return value does not include fields that were read but not assigned. The return value is EOF when the end of the string is encountered before anything is converted.

The value of errno may be set EINVAL, invalid argument.

Example that uses swscanf()

This example uses the swscanf() function to read various data from the string *ltokenstring*, and then displays that data.

```
#include <wchar.h>
#include <stdio.h>
wchar_t *ltokenstring = L"15 12 14";
int i;
float fp;
char s[10];
char c;
int main(void)
   /* Input various data
                                                                    */
  swscanf(ltokenstring, L"%s %c%d%f", s, &c, &i, &fp);
  /* If there were no space between %s and %c,
   /* swscanf would read the first character following */
  /* the string, which is a blank space.
                                                                */
  printf("string = %s\n",s);
  printf("character = %c\n",c);
  printf("integer = %d\n",i);
  printf("floating-point number = %f\n",fp);
```

- "fscanf() Read Formatted Data" on page 113
- "scanf() Read Data" on page 299
- "fwscanf() Read Data from Stream Using Wide Character" on page 128
- "wscanf() Read Data Using Wide-Character Format String" on page 448
- "sscanf() Read Data" on page 322
- "sprintf() Print Formatted Data to Buffer" on page 317
- "<wchar.h>" on page 18

system() — Execute a Command

Format

```
#include <stdlib.h>
int system(const char *string);
```

Language Level: ANSI

Thread Safe: YES. However, the CL command processor and all CL commands are NOT thread safe. Use this function with caution.

Description

The system() function passes the given string to the CL command processor for processing.

Return Value

If passed a non-NULL pointer to a string, the system() function passes the argument to the CL command processor. The system() function returns zero if the command is successful. If passed a NULL pointer to a string, system() returns -1,

and the command processor is not called. If the command fails, system() returns 1. If the system() function fails, the global variable _EXCP_MSGID in <stddef.h> is set with the exception message ID.

Example that uses system()

```
#include <stdlib.h>
int main(void)
{
  int result;

  /* A data area is created, displayed and deleted: */
  result = system("CRTDTAARA QTEMP/TEST TYPE(*CHAR) VALUE('Test')");
  result = system("DSPDTAARA TEST");
  result = system("DLTDTAARA TEST");
```

Related Information

- "exit() End Program" on page 73
- "<stdlib.h>" on page 16

tan() — Calculate Tangent

Format

```
#include <math.h>
double tan(double x);
```

Language Level: ANSI

Description

The tan() function calculates the tangent of x, where x is expressed in radians. If x is too large, a partial loss of significance in the result can occur and sets errno to ERANGE. The value of errno may also be set to EDOM.

Return Value

The tan() function returns the value of the tangent of x.

Example that uses tan()

This example computes x as the tangent of $\pi/4$.

```
#include <math.h>
#include <stdio.h>
int main(void)
  double pi, x;
  pi = 3.1415926;
  x = tan(pi/4.0);
  printf("tan(%lf) is %lf\n", pi/4, x);
/******* Output should be similar to: *********
tan( 0.785398 ) is 1.000000
```

- "acos() Calculate Arccosine" on page 39
- "asin() Calculate Arcsine" on page 43
- "atan() atan2() Calculate Arctangent" on page 45
- "cos() Calculate Cosine" on page 64
- "cosh() Calculate Hyperbolic Cosine" on page 65
- "sin() Calculate Sine" on page 315
- "sinh() Calculate Hyperbolic Sine" on page 316
- "tanh() Calculate Hyperbolic Tangent"
- "<math.h>" on page 8

tanh() — Calculate Hyperbolic Tangent

Format

```
#include <math.h>
double tanh(double x);
```

Language Level: ANSI

Description

The tanh() function calculates the hyperbolic tangent of x, where x is expressed in radians.

Return Value

The tanh() function returns the value of the hyperbolic tangent of x. The result of tanh() cannot have a range error.

Example that uses tanh()

This example computes x as the hyperbolic tangent of $\pi/4$.

- "acos() Calculate Arccosine" on page 39
- "asin() Calculate Arcsine" on page 43
- "atan() atan2() Calculate Arctangent" on page 45
- "cos() Calculate Cosine" on page 64
- "cosh() Calculate Hyperbolic Cosine" on page 65
- "sin() Calculate Sine" on page 315
- "sinh() Calculate Hyperbolic Sine" on page 316
- "tan() Calculate Tangent" on page 371
- "<math.h>" on page 8

time() — Determine Current Time

Format

```
#include <time.h>
time_t time(time_t *timeptr);
```

Language Level: ANSI

Description

The time() function determines the current calendar time, in seconds.

Note:

- Calendar time is the number of seconds that have elapsed since EPOCH, which is 00:00:00, January 1, 1970 Universal Coordinate Time (UTC).
- On the iSeries system, the time function uses the QUTCOFFSET system
 value for calculating UTC. The QUTCOFFSET value specifies the
 difference in hours and minutes between UTC, also known as Greenwich
 mean time, and the current system time. You can override the
 QUTCOFFSET value by specifying the TZDIFF and TZNAME category of
 the current locale.

Return Value

The time() function returns the current calendar time. The return value is also stored in the location that is given by *timeptr*. If *timeptr* is NULL, the return value is not stored. If the calendar time is not available, the value (time t)(-1) is returned.

Example that uses time()

This example gets the time and assigns it to *ltime*. The ctime() function then converts the number of seconds to the current date and time. This example then prints a message giving the current time.

```
#include <time.h>
#include <stdio.h>
int main(void)
  time t ltime;
  if(time(localtime()) == -1)
  printf("Calendar time not available.\n");
  exit(1);
  printf("The time is %s\n", ctime(localtime()));
/******* Output should be similar to: *********
The time is Mon Mar 22 19:01:41 1993
```

Related Information

- "asctime() Convert Time to Character String" on page 40
- "asctime_r() Convert Time to Character String (Restartable)" on page 42
- "ctime() Convert Time to Character String" on page 66
- "ctime_r() Convert Time to Character String (Restartable)" on page 67
- "gmtime() Convert Time" on page 141
- "gmtime_r() Convert Time (Restartable)" on page 143
- "localtime() Convert Time" on page 163
- "localtime_r() Convert Time (Restartable)" on page 164
- "mktime() Convert Local Time" on page 193
- "<time.h>" on page 17

tmpfile() — Create Temporary File

Format

```
#include <stdio.h>
FILE *tmpfile(void);
```

Language Level: ANSI

Description

The tmpfile() function creates a temporary binary file. The file is automatically removed when it is closed or when the program is ended.

The tmpfile() function opens the temporary file in wb+ mode.

Return Value

The tmpfile() function returns a stream pointer, if successful. If it cannot open the file, it returns a NULL pointer. On normal end (exit()), these temporary files are removed.

On the iSeries Data Management system, the tmpfile() function creates a new file that is named QTEMP/QACXxxxx. If you specify the SYSIFCOPT(*IFSIO) option on the compilation command, the tmpfile() function creates a new file that is named /tmp/QACXaaaaaaaa. At the end of the job, the file that is created with the filename from the tmpfile() function is discarded. You can use the remove() function to remove files.

Example that uses tmpfile()

This example creates a temporary file, and if successful, writes tmpstring to it. At program end, the file is removed.

```
#include <stdio.h>
FILE *stream;
char tmpstring[] = "This is the string to be temporarily written";
int main(void)
   if((stream = tmpfile()) == NULL)
     perror("Cannot make a temporary file");
     fprintf(stream, "%s", tmpstring);
```

Related Information

- "fopen() Open Files" on page 92
- "<stdio.h>" on page 15

tmpnam() — Produce Temporary File Name

Format

```
#include <stdio.h>
char *tmpnam(char *string);
```

Language Level: ANSI

Thread Safe: YES. However, using tmpnam(NULL) is NOT thread safe.

Description

The tmpnam() function produces a valid file name that is not the same as the name of any existing file. It stores this name in *string*. If *string* is NULL, the tmpnam() function leaves the result in an internal static buffer. Any subsequent calls destroy this value. If string is not NULL, it must point to an array of at least L_tmpnam bytes. The value of L_tmpnam is defined in <stdio.h>.

The tmpnam() function produces a different name each time it is called within an activation group up to at least TMP_MAX names. For ILE C, TMP_MAX is 32 767. This is a theoretical limit; the actual number of files that can be opened at the same time depends on the available space in the system.

Return Value

The tmpnam() function returns a pointer to the name. If it cannot create a unique name then it returns NULL.

Example that uses tmpnam()

This example calls tmpnam() to produce a valid file name.

```
#include <stdio.h>
int main(void)
  char *name1;
  if ((name1 = tmpnam(NULL)) !=NULL)
     printf("%s can be used as a file name.\n", name1);
  else printf("Cannot create a unique file name\n");
```

Related Information

- "fopen() Open Files" on page 92
- "remove() Delete File" on page 243
- "<stdio.h>" on page 15

toascii() — Convert Character

Format

```
#include <ctype.h>
int toascii(int c);
```

Language Level: XPG4

Description

The toascii() function determines to what character c would be mapped to in a 7-bit US-ASCII locale and returns the corresponding EBCDIC encoding in the current locale.

Return Value

The toascii() function maps the character c according to a 7-bit US-ASCII locale and returns the corresponding EBCDIC encoding in the current locale.

Example that uses toascii()

This example prints EBCDIC encodings of the 7-bit US-ASCII characters 0x7c to 0x82 are mapped to by toascii().

```
#include <stdio.h>
#include <ctype.h>
void main(void)
  int ch;
  for (ch=0x7c; ch<=0x82; ch++) {
   printf("toascii(%#04x) = %c\n", ch, toascii(ch));
/***********And the output should be:**************
toascii(0x7c) = 0
toascii(0x7d) = '
toascii(0x7e) = =
toascii(0x7f) = "
toascii(0x80) = X
toascii(0x81) = a
toascii(0x82) = b
```

- "isascii() Test for ASCII Value" on page 146
- "<ctype.h>" on page 3

tolower() - toupper() - Convert Character Case

Format

```
#include <ctype.h>
int tolower(int C);
int toupper(int c);
```

Language Level: ANSI

Description

The tolower() function converts the uppercase letter C to the corresponding lowercase letter.

The toupper() function converts the lowercase letter *c* to the corresponding uppercase letter.

Return Value

Both functions return the converted character. If the character c does not have a corresponding lowercase or uppercase character, the functions return *c* unchanged.

Note: This function is locale sensitive.

Example that uses toupper() and tolower()

This example uses the toupper() and tolower() functions to change characters between code 0 and code 7f.

```
#include <stdio.h>
#include <ctype.h>
int main(void)
   int ch;
  for (ch = 0; ch \leq 0x7f; ch++)
      printf("toupper=%\#04x\n", toupper(ch));
      printf("tolower=%#04x\n", tolower(ch));
      putchar('\n');
```

- "isalnum() isxdigit() Test Integer Value" on page 147
- "towlower() –towupper() Convert Wide Character Case" on page 379
- "<ctype.h>" on page 3

towctrans() — Translate Wide Character

Format

```
#include <wctype.h>
wint_t towctrans(wint_t wc, wctrans_t desc);
```

Language Level: ANSI

Description

The towctrans() function maps the wide character wc using the mapping that is described by desc. The current setting of the LC_CTYPE category shall be the same as the one used during the call to the towctrans() function that returned the value desc.

A towctrans(wc, wctrans("tolower")) behaves in the same way as the call to the wide-character, case-mapping function towlower().

A towctrans(wc, wctrans("toupper")) behaves in the same way as the call to the wide-character, case-mapping function towupper().

Return Value

The towctrans() function returns the mapped value of wc using the mapping that is described by desc.

Example that uses towctrans()

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <wchar.h>
#include <wctype.h>
int main()
   char *alpha = "abcdefghijklmnopqrstuvwxyz";
   char *tocase[2] = {"toupper", "tolower"};
  wchar_t *wcalpha;
   int i, j;
   size_t alphalen;
   alphalen = strlen(alpha)+1;
   wcalpha = (wchar_t *)malloc(sizeof(wchar_t)*alphalen);
  mbstowcs(wcalpha, alpha, 2*alphalen);
   for (i=0; i<2; ++i) {
      printf("Input string: %ls\n", wcalpha);
      for (j=0; j
      for (j=0; j
```

- "wctrans() —Get Handle for Character Mapping" on page 437
- "<wchar.h>" on page 18

towlower() -towupper() - Convert Wide Character Case

Format

```
#include <wctype.h>
wint_t towlower(wint_t wc);
wint_t towupper(wint_t wc);
```

Thread Safe: YES.

Description

The towupper() function converts the lowercase character wc to the corresponding uppercase letter. The towlower() function converts the uppercase character wc to the corresponding lowercase letter.

Note: This function is accessible only if LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

Return Value

If wc is a wide character for which iswupper() (or iswlower()) is true and there is a corresponding wide character for which iswlower() (or iswupper()) is true, towlower() (or towupper()) returns the corresponding wide character. Otherwise, the argument is returned unchanged.

Example that uses towlower() and towupper()

This example uses towlower() and towupper() to convert characters between 0 and 0x7f.

```
#include <wctvpe.h>
#include <stdio.h>
int main(void)
    wint t w ch;
    for (w_ch = 0; w_ch <= 0xff; w_ch++) {
        printf ("towupper: %#04x %#04x, ", w_ch, towupper(w_ch));
        printf ("towlower : %#04x %#04x\n", w_ch, towlower(w_ch));
    return 0;
The output should be similar to:
towupper: 0xc1 0xc1, towlower: 0xc1 0x81
towupper: 0xc2 0xc2, towlower: 0xc2 0x82
towupper : 0xc3 0xc3, towlower : 0xc3 0x83
towupper: 0xc4 0xc4, towlower: 0xc4 0x84
towupper: 0xc5 0xc5, towlower: 0xc5 0x85
towupper: 0x81 0xc1, towlower: 0x81 0x81
towupper: 0x82 0xc2, towlower: 0x82 0x82
towupper: 0x83 0xc3, towlower: 0x83 0x83
towupper: 0x84 0xc4, towlower: 0x84 0x84
towupper: 0x85 0xc5, towlower: 0x85 0x85
```

- "iswalnum() to iswxdigit() Test Wide Integer Value" on page 150
- "tolower() toupper() Convert Character Case" on page 377
- "<wctype.h>" on page 18

_ultoa - Convert Unsigned Long Integer to String

Format

```
#include <stdlib.h>
char * ultoa(unsigned long value, char *string, int radix);
```

Note: The ultoa function is supported only for C++, not for C.

Language Level: Extension

Description

_ultoa converts the digits of the given unsigned long *value* to a character string that ends with a null character and stores the result in *string*. The *radix* argument specifies the base of *value*; it must be in the range 2 to 36.

Note: The space allocated for *string* must be large enough to hold the returned string. The function can return up to 33 bytes including the null character (\0).

Return Value

_ultoa returns a pointer to *string*. There is no error return value.

Example that uses _ultoa()

This example converts the integer value 255 to a decimal, binary, and hexidecimal representation.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
  char buffer[35];
  char *p;
  p = \_ultoa(255UL, buffer, 10);
  printf("The result of ultoa(255) with radix of 10 is %s\n", p);
  p = ultoa(255UL, buffer, 2);
  printf("The result of _ultoa(255) with radix of 2\n
                                                       is %s\n", p);
  p = ultoa(255UL, buffer, 16);
  printf("The result of ultoa(255) with radix of 16 is %s\n", p);
   return 0;
The output should be:
```

```
The result of ultoa(255) with radix of 10 is 255
The result of _ultoa(255) with radix of 2
   is 11111111
The result of _ultoa(255) with radix of 16 is ff
```

Related Information

- "_gcvt Convert Floating-Point to String" on page 132
- "_itoa Convert Integer to String" on page 154
- "_ltoa Convert Long Integer to String" on page 167
- <stdlib.h>

ungetc() — Push Character onto Input Stream

Format

```
#include <stdio.h>
int ungetc(int c, FILE *stream);
```

Language Level: ANSI

Description

The ungetc() function pushes the unsigned character c back onto the given input stream. However, only one consecutive character is guaranteed to be pushed back onto the input stream if you call ungetc()consecutively. The stream must be open for reading. A subsequent read operation on the stream starts with c. The character c cannot be the EOF character.

Characters placed on the stream by ungetc() will be erased if fseek(), fsetpos(), rewind(), or fflush() is called before the character is read from the *stream*.

Return Value

The ungetc() function returns the integer argument *c* converted to an unsigned char, or EOF if *c* cannot be pushed back.

The value of errno may be set to:

Value Meaning

ENOTREAD

The file is not open for read operations.

EIOERROR

A non-recoverable I/O error occurred.

EIORECERR

A recoverable I/O error occurred.

The ungetc() function is not supported for files opened with type=record.

Example that uses ungetc()

In this example, the while statement reads decimal digits from an input data stream by using arithmetic statements to compose the numeric values of the numbers as it reads them. When a non-digit character appears before the end of the file, ungetc() replaces it in the input stream so that later input functions can process it.

```
#include <stdio.h>
#include <ctype.h>
int main(void)
  FILE *stream;
  int ch;
  unsigned int result = 0;
  while ((ch = getc(stream)) != EOF && isdigit(ch))
      result = result * 10 + ch - '0';
  if (ch != EOF)
     ungetc(ch, stream);
         /* Put the nondigit character back */
  printf("The result is: %d\n", result);
  if ((ch = getc(stream)) != EOF)
     printf("The character is: %c\n", ch);
```

Related Information

- "getc() getchar() Read a Character" on page 133
- "fflush() Write Buffer to File" on page 80
- "fseek() fseeko() Reposition File Position" on page 114
- "fsetpos() Set File Position" on page 117
- "putc() putchar() Write a Character" on page 210
- "rewind() Adjust Current File Position" on page 245
- "<stdio.h>" on page 15

ungetwc() — Push Wide Character onto Input Stream

Format

```
#include <wchar.h>
#include <stdio.h>
wint_t ungetwc(wint_t wc, FILE *stream);
```

Language Level: ANSI

Description

The ungetwc() function pushes the wide character wc back onto the input stream. The pushed-back wide characters will be returned by subsequent reads on that stream in the reverse order of their pushing. A successful intervening call (on the stream) to a file positioning function (fseek(), fsetpos(), or rewind()) discards any pushed-back wide characters for the stream. The external storage corresponding to the stream is unchanged. There is always at least one wide character of push-back. If the value of wc is WEOF, the operation fails and the input stream is unchanged.

A successful call to the ungetwc() function clears the EOF indicator for the stream. The value of the file position indicator for the stream after reading or discarding all pushed-back wide characters is the same as it was before the wide characters were pushed back. However, only one consecutive wide character is guaranteed to be pushed back onto the input stream if you call ungetwc() consecutively.

For a text stream, the file position indicator is backed up by one wide character. This affects the ftell(), fflush(), fseek() (with SEEK_CUR), and fgetpos()function. For a binary stream, the position indicator is unspecified until all characters are read or discarded, unless the last character is pushed back, in which case the file position indicator is backed up by one wide character. This affects the ftell(), fseek() (with SEEK_CUR), fgetpos(), and fflush()function.

Note: This function is available when SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) are specified on the compilation command.

Return Value

The ungetwc() function returns the wide character pushed back after conversion, or WEOF if the operation fails.

Example that uses ungetwc()

```
#include <wchar.h>
#include <wctvpe.h>
#include <stdio.h>
#include <stdlib.h>
int main(void)
  FILE
                *stream:
  wint t
               WC;
  wint_t
               wc2;
  unsigned int result = 0;
   if (NULL == (stream = fopen("ungetwc.dat", "r+"))) {
     printf("Unable to open file.\n");
     exit(EXIT FAILURE);
  while (WEOF != (wc = fgetwc(stream)) && iswdigit(wc))
     result = result * 10 + wc - L'0';
   if (WEOF != wc)
     ungetwc(wc, stream);
                             /* Push the nondigit wide character back */
   /* get the pushed back character */
   if (WEOF != (wc2 = fgetwc(stream))) {
     if (wc != wc2) {
        printf("Subsequent fgetwc does not get the pushed back character.\n");
         exit(EXIT FAILURE);
```

```
printf("The digits read are '%i'\n"
       "The character being pushed back is '%1c'", result, wc2);
return 0;
Assuming the file ungetwo.dat contains:
  12345ABCDE67890XYZ
  The output should be similar to:
  The digits read are '12345'
  The character being pushed back is 'A'
                              ***********
```

- "fflush() Write Buffer to File" on page 80
- "fseek() fseeko() Reposition File Position" on page 114
- "fsetpos() Set File Position" on page 117
- "getwc() Read Wide Character from Stream" on page 138
- "putwc() Write Wide Character" on page 213
- "ungetc() Push Character onto Input Stream" on page 381
- "<wchar.h>" on page 18

va_arg() - va_end() - va_start() — Access Function Arguments

Format

```
#include <stdarg.h>
var type va arg(va list arg_ptr, var_type);
void va_end(va_list arg_ptr);
void va_start(va_list arg_ptr, variable_name);
```

Language Level: ANSI

Description

The va_arg(), va_end(), and va_start() functions access the arguments to a function when it takes a fixed number of required arguments and a variable number of optional arguments. You declare required arguments as ordinary parameters to the function and access the arguments through the parameter names.

va_start() initializes the *arg_ptr* pointer for subsequent calls to va_arg() and va end().

The argument variable name is the identifier of the rightmost named parameter in the parameter list (preceding, ...). Its type must be one of int, long, decimal, double, struct, union, or pointer, or a typedef of one of these types. Use va start() before va_arg(). Corresponding va_start() and va_end() macros must be in the same function.

The va arg() function retrieves a value of the given var type from the location given by arg_ptr, and increases arg_ptr to point to the next argument in the list. The va arg() function can retrieve arguments from the list any number of times within

the function. The *var_type* argument must be one of int, long, decimal, double, struct, union, or pointer, or a typedef of one of these types.

The va_end() function is needed to indicate the end of parameter scanning.

Return Value

The va_arg() function returns the current argument. The va_end and va_start() functions do not return a value.

```
Example that uses va_arg() - va_end() - va_start()
```

This example passes a variable number of arguments to a function, stores each argument in an array, and prints each argument.

```
#include <stdio.h>
#include <stdarg.h>
int vout(int max, ...);
int main(void)
  vout(3, "Sat", "Sun", "Mon");
  printf("\n");
  vout(5, "Mon", "Tues", "Wed", "Thurs", "Fri");
int vout(int max, ...)
  va list arg ptr;
  int args = 0;
  char *days[7];
  va_start(arg_ptr, max);
  while(args < max)</pre>
     days[args] = va_arg(arg_ptr, char *);
     printf("Day: %s \n", days[args++]);
  va end(arg ptr);
/****** Output should be similar to: ********
Day: Sat
Day: Sun
Day: Mon
Day: Mon
Day: Tues
Day: Wed
Day: Thurs
Day: Fri
```

- "vfprintf() Print Argument Data to Stream" on page 386
- "vprintf() Print Argument Data" on page 389
- "vfwprintf() Format Argument Data as Wide Characters and Write to a Stream" on page 387
- "vsprintf() Print Argument Data to Buffer" on page 391

vfprintf() — Print Argument Data to Stream

Format

```
#include <stdarg.h>
#include <stdio.h>
int vfprintf(FILE *stream, const char *format, va_list arg_ptr);
```

Language Level: ANSI

Description

The vfprintf() function formats and writes a series of characters and values to the output *stream*. The vfprintf() function works just like the fprintf() function, except that *arg_ptr* points to a list of arguments whose number can vary from call to call in the program. These arguments should be initialized by va_start for each call. In contrast, the fprintf() function can have a list of arguments, but the number of arguments in that list is fixed when you compile the program.

The vfprintf() function converts each entry in the argument list according to the corresponding format specifier in *format*. The *format* has the same form and function as the format string for the printf() function.

Return Value

If successful, vfprintf() returns the number of bytes written to *stream*. If an error occurs, the function returns a negative value.

Example that uses vfprintf()

This example prints out a variable number of strings to the file myfile.

```
#include <stdarg.h>
#include <stdio.h>
void vout(FILE *stream, char *fmt, ...);
char fmt1 [] = %s %s %s\n";
int main(void)
  FILE *stream;
  stream = fopen("mylib/myfile", "w");
  vout(stream, fmt1, "Sat", "Sun", "Mon");
void vout(FILE *stream, char *fmt, ...)
  va list arg ptr;
  va_start(arg_ptr, fmt);
  vfprintf(stream, fmt, arg ptr);
  va_end(arg_ptr);
/******* Output should be similar to: *********
Sat Sun Mon
*/
```

- "fprintf() Write Formatted Data to a Stream" on page 99
- "printf() Print Formatted Characters" on page 200
- "va_arg() va_end() va_start() Access Function Arguments" on page 384
- "vprintf() Print Argument Data" on page 389
- "vsprintf() Print Argument Data to Buffer" on page 391
- "vwprintf() Format Argument Data as Wide Characters and Print" on page 394
- "<stdarg.h>" on page 14
- "<stdio.h>" on page 15

vfwprintf() — Format Argument Data as Wide Characters and Write to a Stream

Format

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>
int vfwprintf(FILE *stream, const wchar_t *format, va_list arg);
```

Language Level: ANSI

Description

The vfwprintf() function is equivalent to the fwprintf() function, except that the variable argument list is replaced by *arg*, which the va_start macro (and possibly subsequent va_arg calls) will have initialized. The vfwprintf() function does not invoke the va_end macro.

Because the functions vfwprintf(), vswprintf(), and vwprintf()invoke the va_arg macro, the value of arg after the return is unspecified.

Note: This function is available when SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALE) are specified on the compilation command.

Return Value

The vfwprintf() function returns the number of wide characters transmitted. If an output error occurred, it returns a negative value.

Example that uses vfwprintf()

This example prints the wide character a to a file.. The printing is done from the vout () function, which takes a variable number of arguments and uses vfwprintf() to print them to a file.

```
#include <wchar.h>
#include <stdarg.h>
#include <locale.h>
void vout (FILE *stream, wchar t *fmt, ...);
const char ifs path [] = "tmp/myfile";
int main(void) {
 FILE *stream;
 wchar_t format [] = L"%1c";
 setlocale(LC ALL, "POSIX");
 if ((stream = fopen (ifs path, "w")) == NULL) {
   printf("Could not open file.\n");
   return (-1);
 vout (stream, format, L'a');
 fclose (stream);
 /**************
   The contents of output file tmp/myfile.dat should
   be a wide char 'a' which in the "POSIX" locale
   is '0081'x.
 return (0);
void vout (FILE *stream, wchar t *fmt, ...)
 va_list arg_ptr;
  va_start (arg_ptr, fmt);
  vfwprintf (stream, fmt, arg ptr);
 va_end (arg_ptr);
```

- "printf() Print Formatted Characters" on page 200
- "fprintf() Write Formatted Data to a Stream" on page 99
- "vfprintf() Print Argument Data to Stream" on page 386
- "vprintf() Print Argument Data" on page 389

- "btowc() Convert Single Byte to Wide Character" on page 54
- "mbrtowc() Convert a Multibyte Character to a Wide Character (Restartable)" on page 176
- "fwprintf() Format Data as Wide Characters and Write to a Stream" on page 123
- "vswprintf() Format and Write Wide Characters to Buffer" on page 393
- "vwprintf() Format Argument Data as Wide Characters and Print" on page 394
- "<stdarg.h>" on page 14
- "<stdio.h>" on page 15
- "<wchar.h>" on page 18

vprintf() — Print Argument Data

Format

```
#include <stdarg.h>
#include <stdio.h>
int vprintf(const char *format, va list arg ptr);
```

Language Level: ANSI

Description

The vprintf() function formats and prints a series of characters and values to stdout. The vprintf() function works just like the printf() function, except that *arg_ptr* points to a list of arguments whose number can vary from call to call in the program. These arguments should be initialized by va_start for each call. In contrast, the printf() function can have a list of arguments, but the number of arguments in that list is fixed when you compile the program.

The vprintf() function converts each entry in the argument list according to the corresponding format specifier in *format*. The *format* has the same form and function as the format string for the printf() function.

Return Value

If successful, the vprintf() function returns the number of bytes written to stdout. If an error occurs, the vprintf() function returns a negative value. The value of errno may be set to ETRUNC.

Example that uses vprintf()

This example prints out a variable number of strings to stdout.

```
#include <stdarg.h>
#include <stdio.h>
void vout(char *fmt, ...);
char fmt1 [] = "%s %s %s %s %s \n";
int main(void)
  FILE *stream;
  stream = fopen("mylib/myfile", "w");
  vout(fmt1, "Mon", "Tues", "Wed", "Thurs", "Fri");
void vout(char *fmt, ...)
  va list arg ptr;
  va_start(arg_ptr, fmt);
  vprintf(fmt, arg_ptr);
  va end(arg ptr);
/******* Output should be similar to: ********
Mon Tues Wed Thurs
                       Fri
```

- "printf() Print Formatted Characters" on page 200
- "va_arg() va_end() va_start() Access Function Arguments" on page 384
- "vfprintf() Print Argument Data to Stream" on page 386
- "vsprintf() Print Argument Data to Buffer" on page 391
- "<stdarg.h>" on page 14
- "<stdio.h>" on page 15

vsnprintf() — Print Argument Data to Buffer

Format

```
#include <stdarg.h>
#include <stdio.h>
int vsnprintf(char *target-string, size t n, const char *format, va list arg ptr);
```

Language Level: ANSI

Description

The vsnprintf() function formats and stores a series of characters and values in the buffer target-string. The vsnprintf() function works just like the snprintf() function, except that arg_ptr points to a list of arguments whose number can vary from call to call in the program. These arguments should be initialized by the va start function for each call. In contrast, the snprintf() function can have a list of arguments, but the number of arguments in that list is fixed when you compile the program.

The vsnprintf() function converts each entry in the argument list according to the corresponding format specifier in format. The format has the same form and function as the format string for the printf() function.

Return Value

The vsnprintf() function returns the number of bytes that are written in the array, not counting the ending null character.

Example that uses vsnprintf()

This example assigns a variable number of strings to *string* and prints the resultant string.

Related Information

- "printf() Print Formatted Characters" on page 200
- "sprintf() Print Formatted Data to Buffer" on page 317
- "snprintf() Print Formatted Data to Buffer" on page 320
- "va_arg() va_end() va_start() Access Function Arguments" on page 384
- "vfprintf() Print Argument Data to Stream" on page 386
- "vsprintf() Print Argument Data to Buffer"
- "<stdarg.h>" on page 14
- "<stdio.h>" on page 15

vsprintf() — Print Argument Data to Buffer

Format

```
#include <stdarg.h>
#include <stdio.h>
int vsprintf(char *target-string, const char *format, va_list arg_ptr);
```

Language Level: ANSI

Description

The vsprintf() function formats and stores a series of characters and values in the buffer target-string. The vsprintf() function works just like the sprintf() function, except that arg_ptr points to a list of arguments whose number can vary from call to call in the program. These arguments should be initialized by the va_start function for each call. In contrast, the sprintf() function can have a list of arguments, but the number of arguments in that list is fixed when you compile the program.

The vsprintf() function converts each entry in the argument list according to the corresponding format specifier in format. The format has the same form and function as the format string for the printf() function.

Return Value

If successful, the vsprintf() function returns the number of bytes written to target-string. If an error occurs, the vsprintf() function returns a negative value.

Example that uses vsprintf()

This example assigns a variable number of strings to string and prints the resultant string.

```
#include <stdarg.h>
#include <stdio.h>
void vout(char *string, char *fmt, ...);
char fmt1 [] = "%s %s %s\n";
int main(void)
  char string[100];
  vout(string, fmt1, "Sat", "Sun", "Mon");
  printf("The string is: %s\n", string);
void vout(char *string, char *fmt, ...)
  va list arg ptr;
  va start(arg ptr, fmt);
  vsprintf(string, fmt, arg_ptr);
  va end(arg ptr);
/****** Output should be similar to: *********
The string is: Sat Sun Mon
```

- "printf() Print Formatted Characters" on page 200
- "sprintf() Print Formatted Data to Buffer" on page 317
- "va_arg() va_end() va_start() Access Function Arguments" on page 384
- "vfprintf() Print Argument Data to Stream" on page 386
- "vprintf() Print Argument Data" on page 389
- "vswprintf() Format and Write Wide Characters to Buffer" on page 393
- "<stdarg.h>" on page 14

vswprintf() — Format and Write Wide Characters to Buffer

Format

Language Level: ANSI

Thread Safe: YES.

Description

The vswprintf() function formats and stores a series of wide characters and values in the buffer <code>wcsbuffer</code>. The vswprintf() function works just like the swprintf() function, except that <code>argptr</code> points to a list of wide-character arguments whose number can vary from call to call. These arguments should be initialized by va_start for each call. In contrast, the <code>swprintf()</code> function can have a list of arguments, but the number of arguments in that list are fixed when you compile in the program.

The value n specifies the maximum number of wide characters to be written, including the ending null character. The vswprintf() function converts each entry in the argument list according to the corresponding wide-character format specifier in format. The format has the same form and function as the format string for the printf() function, with the following exceptions:

- %c (without an l prefix) converts an integer argument to wchar_t, as if by calling the mbtowc() function.
- %lc converts a wint_t to wchar_t.
- %s (without an l prefix) converts an array of multibyte characters to an array of wchar_t, as if by calling the mbrtowc() function. The array is written up to, but not including, the ending null character, unless the precision specifies a shorter output.
- %ls writes an array of wchar_t. The array is written up to, but not including, the ending null character, unless the precision specifies a shorter output.

A null wide character is added to the end of the wide characters written; the null wide character is not counted as part of the returned value. If copying takes place between objects that overlap, the behavior is undefined.

Note: This function is available when SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALE) are specified on the compilation command.

Return Value

The vswprintf() function returns the number of bytes written in the array, not counting the ending null wide character.

Example that uses vswprintf()

This example creates a function vout () that takes a variable number of wide-character arguments and uses vswprintf() to print them to wcstr.

```
#include <stdio.h>
#include <stdarg.h>
#include <wchar.h>
wchar t *format3 = L"%ls %d %ls";
wchar t *format5 = L"%ls %d %ls %d %ls";
void vout(wchar_t *wcs, size_t n, wchar_t *fmt, ...)
  va list arg ptr;
  va_start(arg_ptr, fmt);
  vswprintf(wcs, n, fmt, arg ptr);
  va end(arg ptr);
  return;
int main(void)
  wchar t wcstr[100];
  vout(wcstr, 100, format3, L"ONE", 2L, L"THREE");
  printf("%ls\n", wcstr);
  vout(wcstr, 100, format5, L"ONE", 2L, L"THREE", 4L, L"FIVE");
  printf("%1s\n", wcstr);
  return 0;
  /*******************
     The output should be similar to:
     ONE 2 THREE
     ONE 2 THREE 4 FIVE
  *************************************
```

Related Information

- "swprintf() Format and Write Wide Characters to Buffer" on page 367
- "vfprintf() Print Argument Data to Stream" on page 386
- "vprintf() Print Argument Data" on page 389
- "vsprintf() Print Argument Data to Buffer" on page 391
- "<stdarg.h>" on page 14
- "<wchar.h>" on page 18

vwprintf() — Format Argument Data as Wide Characters and Print

```
Format
```

```
#include <stdarg.h>
#include <wchar.h>
int vwprintf(const wchar t *format, va list arg);
```

Language Level: ANSI

Thread Safe: YES.

Description

The vwprintf() function is equivalent to the wprintf() function, except that the variable argument list is replaced by *arg*, which the va_start macro (and possibly subsequent va_arg calls) will have initialized. The vwprintf() function does not invoke the va_end macro.

Note: This function is available when SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALE) are specified on the compilation command.

Return Value

The vwprintf() function returns the number of wide characters transmitted. If an output error occurred, thevwprintf() returns a negative value.

Example that uses vwprintf()

This example prints the wide character *a*. The printing is done from the vout() function, which takes a variable number of arguments and uses the vwprintf()function to print them to stdout.

```
#include <wchar.h>
#include <stdarg.h>
#include <locale.h>
void vout (wchar_t *fmt, ...);
const char ifs path[] = "tmp/mytest";
int main(void)
FILE *stream;
wchar_t format[] = L"%1c";
setlocale(LC_ALL, "POSIX");
vout (format, L'a');
return(0);
/* A long a is written to stdout, if stdout is written to the screen
   it may get converted back to a single byte 'a'. */
void vout (wchar t *fmt, ...) {
va_list arg_ptr;
va_start (arg_ptr, fmt);
vwprintf (fmt, arg_ptr);
va end (arg ptr);
```

- "printf() Print Formatted Characters" on page 200
- "vfprintf() Print Argument Data to Stream" on page 386
- "vprintf() Print Argument Data" on page 389
- "btowc() Convert Single Byte to Wide Character" on page 54
- "mbrtowc() Convert a Multibyte Character to a Wide Character (Restartable)" on page 176
- "fwprintf() Format Data as Wide Characters and Write to a Stream" on page 123
- "vswprintf() Format and Write Wide Characters to Buffer" on page 393
- "vfwprintf() Format Argument Data as Wide Characters and Write to a Stream" on page 387

- "<stdarg.h>" on page 14
- "<wchar.h>" on page 18

wcrtomb() — Convert a Wide Character to a Multibyte Character (Restartable)

Format

#include <wchar.h> size t wcrtomb (char *s, wchar t wc, mbstate t *ps);

Language Level: ANSI

Thread Safe: Yes, except when *ps* is NULL.

Description

This function is the restartable version of the wctomb() function.

The wcrtomb() function converts a wide character to a multibyte character.

If s is a null pointer, the wortomb() function determines the number of bytes necessary to enter the initial shift state (zero if encodings are not state-dependent or if the initial conversion state is described). The resulting state described will be the initial conversion stated.

If s is not a null pointer, the wortomb() function determines the number of bytes needed to represent the multibyte character that corresponds to the wide character given by wc (including any shift sequences), and stores the resulting bytes in the array whose first element is pointed to by s. At most MB CUR MAX bytes will be stored. If wc is a null wide character, the resulting state described will be the initial conversions state.

This function differs from its corresponding internal-state multibyte character function in that it has an extra parameter, ps of type pointer to mbstate_t that points to an object that can completely describe the current conversion state of the associated multibyte character sequence. If ps is NULL, an internal static variable will be used to keep track of the conversion state. Using the internal static variable is not thread safe.

When the program is compiled with LOCALETYPE(*LOCALE) and SYSIFCOPT(*IFSIO), the behavior of wcrtomb() is affected by the LC_CTYPE category of the current locale. Remember that the CCSID of the locale should match the CCSID of your job. If the CCSID of the locale is a single-byte CCSID, the wide characters are converted to single-byte characters. Any wide character whose value is greater than 256 would be invalid when converting to a single-byte CCSID. When the CCSID is a multibyte CCSID, the wide characters are converted to multibyte characters.

When the program is compiled with LOCALETYPE(*LOCALEUCS2) and SYSIFCOPT(*IFSIO), the wide characters are assumed to be UNICODE characters. These UNICODE characters are converted to the CCSID of the locale associated with LC CTYPE.

Return Value

If s is a null pointer, the wcrtomb() function returns the number of bytes needed to enter the initial shift state. The value returned will not be greater than that of the MB_CUR_MAX macro.

If s is not a null pointer, the wcrtomb() function returns the number of bytes stored in the array object (including any shift sequences) when wc is a valid wide character; otherwise (when wc is not a valid wide character), an encoding error occurs, the value of the macro EILSEQ shall be stored in errno and -1 will be returned, but the conversion state will be unchanged.

Examples that use wcrtomb()

This program is compiled with LOCALETYPE(*LOCALE) and SYSIFCOPT(*IFSIO):

```
#include <stdio.h>
#include <locale.h>
#include <wchar.h>
#include <errno.h>
#define STRLENGTH 10
#define LOCNAME
                    "gsys.lib/JA JP.locale"
#define LOCNAME_EN "qsys.lib/EN_US.locale"
int main(void)
             string[STRLENGTH];
   char
   int length, s1 = 0;
   wchar_t wc = 0x4171;
   wchar t wc2 = 0x00C1;
   wchar t wc string[10];
   mbstate t ps = 0;
   memset(string, '\0', STRLENGTH);
   wc_string[0] = 0x00C1;
   wc string[1] = 0x4171;
   wc string[2] = 0x4172;
   wc string[3] = 0x00C2;
   wc string[4] = 0x0000;
   /* In this first example we will convert a wide character */
   /* to a single byte character. We first set the locale */
   /* to a single byte locale. We choose a locale with
   /* CCSID 37. For single byte cases the state will always */
   /* remain in the initial state 0
   if (setlocale(LC ALL, LOCNAME EN) == NULL)
       printf("setlocale failed.\n");
   length = wcrtomb(string, wc, &ps);
   /* In this case since wc > 256 hex, lenth is -1 and */
    /* errno is set to EILSEQ (3492) */
   printf("errno = %d, length = %d\n\n", errno, length);
   length = wcrtomb(string, wc2, &ps);
   /* In this case wc2 00C1 is converted to C1 */
   printf("string = %s\n\n", string);
   /* Now lets try a multibyte example. We first must set the */
   /st locale to a multibyte locale. We choose a locale with st/
   /* CCSID 5026 */
   if (setlocale(LC ALL, LOCNAME) == NULL)
       printf("setlocale failed.\n");
```

```
length = wcrtomb(string, wc string[0], &ps);
    /* The first character is < 256 hex so is converted to */
    /* single byte and the state is still the initial state 0 */
    printf("length = %d, state = %d\n\n", length, ps);
    sl += length;
    length = wcrtomb(&string[s1], wc_string[1], &ps);
    /* The next character is > 256 hex so we get a shift out
    /* 0x0e followed by the double byte character. State is
    /* changed to double byte state. Length is 3.
   printf("length = %d, state = %d\n\n", length, ps);
    sl += length;
    length = wcrtomb(&string[s1], wc string[2], &ps);
    /* The next character is > 256 hex so we get another
   /* double byte character. The state is left in
    /* double byte state. Length is 2.
   printf("length = %d, state = %d\n\n", length, ps);
    sl += length;
    length = wcrtomb(&string[s1], wc_string[3], &ps);
    /* The next character is < 256 hex so we close off the
    /* double byte characters with a shift in 0x0f and then
    /* get a single byte character. Length is 2.
    /* The hex look at string would now be:
    /* C10E417141720FC2
    /* You would need a device capable of displaying multibyte */
    /* characters to see this string.
    printf("length = %d, state = %d\n\n", length, ps);
    /* In the last example we will show what happens if NULL
    /* is passed in for the state.
   memset(string, '\0', STRLENGTH);
    length = wcrtomb(string, wc string[1], NULL);
    /* The second character is > 256 hex so a shift out
    /* followed by the double character is produced but since */
   /* the state is NULL, the double byte character is closed */
   /* off with a shift in right away. So string we look
    /* like this: OE41710F and length is 4 and the state is
    /* left in the initial state.
    printf("length = %d, state = %d\n\n", length, ps);
/* The output should look like this:
errno = 3492, length = -1
string = A
length = 1, state = 0
length = 3, state = 2
```

```
length = 2, state = 2
length = 2, state = 0
length = 4, state = 0
                                  */
This program is compiled with LOCALETYPE(*LOCALEUCS2) and
SYSIFCOPT(*IFSIO):
#include <stdio.h>
#include <locale.h>
#include <wchar.h>
#include <errno.h>
#define STRLENGTH
#define LOCNAME
                    "qsys.lib/JA JP.locale"
#define LOCNAME EN "qsys.lib/EN US.locale"
int main(void)
    char
             string[STRLENGTH];
   int length, sl = 0;
   wchar t wc = 0x4171:
   wchar t wc2 = 0x0041;
   wchar_t wc_string[10];
   mbstate_t ps = 0;
   memset(string, '\0', STRLENGTH);
   wc_string[0] = 0x0041;
   wc_string[1] = 0xFF31;
   wc string[2] = 0xFF32;
   wc string[3] = 0x0042;
   wc string[4] = 0x0000;
   /* In this first example we will convert a UNICODE character */
   /* to a single byte character. We first set the locale */
   /* to a single byte locale. We choose a locale with
    /* CCSID 37. For single byte cases the state will always */
    /* remain in the initial state 0
   if (setlocale(LC ALL, LOCNAME EN) == NULL)
       printf("setlocale failed.\n");
   length = wcrtomb(string, wc2, &ps);
    /* In this case wc2 0041 is converted to C1 */
    /* 0041 is UNICODE A, C1 is CCSID 37 A
   printf("string = %s\n\n", string);
   /* Now lets try a multibyte example. We first must set the */
   /* locale to a multibyte locale. We choose a locale with
   /* CCSID 5026 */
   if (setlocale(LC ALL, LOCNAME) == NULL)
       printf("setlocale failed.\n");
    length = wcrtomb(string, wc string[0], &ps);
    /* The first character UNICODE character is converted to a */
    /* single byte and the state is still the initial state 0 */
   printf("length = %d, state = %d\n\n", length, ps);
   sl += length;
   length = wcrtomb(&string[s1], wc string[1], &ps);
```

```
/* The next UNICODE character is converted to a shift out
    /* 0x0e followed by the double byte character. State is
    /* changed to double byte state. Length is 3.
    printf("length = %d, state = %d\n\n", length, ps);
    sl += length;
    length = wcrtomb(&string[s1], wc_string[2], &ps);
    /* The UNICODE character is converted to another
    /* double byte character. The state is left in
    /* double byte state. Length is 2.
    printf("length = %d, state = %d\n\n", length, ps);
   s1 += length;
   length = wcrtomb(&string[s1], wc string[3], &ps);
    /* The next UNICODE character converts to single byte so
    /* we close off the
    /* double byte characters with a shiftin 0x0f and then
    /* get a single byte character. Length is 2.
    /* The hex look at string would now be:
    /* C10E42D842D90FC2
    /* You would need a device capable of displaying multibyte */
    /* characters to see this string.
    printf("length = %d, state = %d\n\n", length, ps);
/* The output should look like this:
string = A
length = 1, state = 0
length = 3, state = 2
length = 2, state = 2
length = 2, state = 0
                                   */
```

- "mblen() Determine Length of a Multibyte Character" on page 172
- "mbrlen() Determine Length of a Multibyte Character (Restartable)" on page 173
- "mbrtowc() Convert a Multibyte Character to a Wide Character (Restartable)" on page 176
- "mbsrtowcs() Convert a Multibyte String to a Wide Character String (Restartable)" on page 180
- "wcsrtombs() Convert Wide Character String to Multibyte String (Restartable)" on page 418
- "wctomb() Convert Wide Character to Multibyte Character" on page 436
- "<wchar.h>" on page 18

wcscat() — Concatenate Wide-Character Strings

Format

```
#include <wcstr.h>
wchar t *wcscat(wchar t *string1, const wchar t *string2);
```

Language Level: XPG4

Description

The wcscat() function appends a copy of the string pointed to by *string2* to the end of the string pointed to by *string1*.

The wcscat() function operates on null-ended wchar_t strings. The string arguments to this function should contain a wchar_t null character marking the end of the string. Boundary checking is not performed.

Return Value

The wcscat() function returns a pointer to the concatenated *string1*.

Example that uses wcscat()

This example creates the wide character string "computer program" using the wcscat() function.

- "strcat() Concatenate Strings" on page 324
- "strncat() Concatenate Strings" on page 343
- "wcschr() Search for Wide Character" on page 402
- "wcscmp() Compare Wide-Character Strings" on page 403
- "wcscpy() Copy Wide-Character Strings" on page 405
- "wcscspn() Find Offset of First Wide-Character Match" on page 406
- "wcslen() Calculate Length of Wide-Character String" on page 409
- "wcsncat() Concatenate Wide-Character Strings" on page 410
- "<wcstr.h>" on page 18

wcschr() — Search for Wide Character

Format

```
#include <wcstr.h>
wchar_t *wcschr(const wchar_t *string, wchar_t character);
```

Language Level: XPG4

Description

The wcschr() function searches the wide-character string for the occurrence of *character*. The *character* can be a wchar_t null character (\0); the wchar_t null character at the end of string is included in the search.

The wcschr() function operates on null-ended wchar_t strings. The string argument to this function should contain a wchar_t null character marking the end of the string.

Return Value

The wcschr() function returns a pointer to the first occurrence of *character* in *string*. If the character is not found, a NULL pointer is returned.

Example that uses wcschr()

This example finds the first occurrence of the character "p" in the wide-character string "computer program".

```
#include <stdio.h>
#include <wcstr.h>
#define SIZE 40
int main(void)
 wchar t buffer1[SIZE] = L"computer program";
 wchar_t * ptr;
 wchar_t ch = L'p';
 ptr = wcschr( buffer1, ch );
 printf( "The first occurrence of %lc in '%ls' is '%ls'\n",
                        ch, buffer1, ptr );
/****** Output should be similar to: *********
The first occurrence of p in 'computer program' is 'puter program'
```

- "strchr() Search for Character" on page 325
- "strcspn() Find Offset of First Character Match" on page 331
- "strpbrk() Find Characters in String" on page 348
- "strrchr() Locate Last Occurrence of Character in String" on page 354
- "strspn() —Find Offset of First Non-matching Character" on page 355
- "wcscat() Concatenate Wide-Character Strings" on page 400
- "wcscmp() Compare Wide-Character Strings" on page 403

- "wcscpy() Copy Wide-Character Strings" on page 405
- "wcscspn() Find Offset of First Wide-Character Match" on page 406
- "wcslen() Calculate Length of Wide-Character String" on page 409
- "wcsncmp() Compare Wide-Character Strings" on page 411
- "wcspbrk() Locate Wide Characters in String" on page 416
- "wcsrchr() Locate Last Occurrence of Wide Character in String" on page 417
- "wcsspn() Find Offset of First Non-matching Wide Character" on page 420
- "wcswcs() Locate Wide-Character Substring" on page 432
- "<wcstr.h>" on page 18

wcscmp() — Compare Wide-Character Strings

Format

I

```
#include <wcstr.h>
int wcscmp(const wchar_t *string1, const wchar_t *string2);
```

Language Level: ANSI

Description

The wcscmp() function compares two wide-character strings. The wcscmp()function operates on null-ended wchar_t strings; string arguments to this function should contain a wchar_t null character marking the end of the string. Boundary checking is not performed when a string is added to or copied.

Return Value

The wcscmp() function returns a value indicating the relationship between the two strings, as follows:

Value Meaning

Less than 0

string1 less than string2

0 *string1* identical to *string2*

Greater than 0

string1 greater than string2.

Example that uses wcscmp()

This example compares the wide-character string *string1* to *string2* using wcscmp().

```
#include <stdio.h>
#include <wcstr.h>

int main(void)
{
   int result;
   wchar_t string1[] = L"abcdef";
   wchar_t string2[] = L"abcdefg";

   result = wcscmp( string1, string2 );

   if ( result == 0 )
        printf( "\"%1s\" is identical to \"%1s\"\n", string1, string2);
   else if ( result < 0 )
        printf( "\"%1s\" is less than \"%1s\"\n", string1, string2 );</pre>
```

```
else
   printf( "\"%ls\" is greater than \"%ls\"\n", string1, string2);
/****** Output should be similar to: *********
"abcdef" is less than "abcdefg"
```

- "strcmp() Compare Strings" on page 326
- "strncmp() Compare Strings" on page 344
- "wcscat() Concatenate Wide-Character Strings" on page 400
- "wcschr() Search for Wide Character" on page 402
- "wcscpy() Copy Wide-Character Strings" on page 405
- "wcscspn() Find Offset of First Wide-Character Match" on page 406
- "wcslen() Calculate Length of Wide-Character String" on page 409
- "wcsncmp() Compare Wide-Character Strings" on page 411
- "_wcsicmp Compare Wide Character Strings without Case Sensitivity" on page 413
- "_wcsnicmp Compare Wide Character Strings without Case Sensitivity" on page 415
- "<wcstr.h>" on page 18

wcscoll() —Language Collation String Comparison

Format

```
#include <wchar.h>
int wcscoll (const wchar_t *wcs1, const wchar_t *wcs2);
```

Language Level: XPG4

Description

The wcscoll() function compares the wide-character strings pointed to by wcs1 and wcs2, both interpreted as appropriate to the LC_COLLATE category of the current locale.

Note: This function is accessible only if LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

The behavior of this wide-character function is affected by the LC_COLLATE category of the current locale.

Although this function is accessible when compiled with LOCALETYPE(*LOCALEUCS2), UNICODE is not supported by this function.

Return Value

The wcscoll() function returns an integer value indicationg the relationship between the strings, as follows:

Value Meaning

Less than 0

wcs1 less than wcs2

0 *wcs1* equivalent to *wcs2*

Greater than 0

wcs1 greater than wcs2

If *wcs1* or *wcs2* contain characters outside the domain of the collating sequence, the wcscoll() function sets errno to EINVAL. If an error occurs, the wcscoll() function sets errno to an nonzero value. There is no error return value.

If wcscoll() is unsuccessful, errno is changed. The value of errno may be set to EINVAL (the *wcs1* or *wcs2* arguments contain characters which are not available in the current locale).

Example that uses wcscoll()

This example uses the default locale.

```
#include <stdio.h>
#include <wchar.h>

int main(void)
{
    int result;
    wchar_t *wcs1 = L"first_wide_string";
    wchar_t *wcs2 = L"second_wide_string";

    result = wcscoll(wcs1, wcs2);

    if ( result == 0)
        printf("\"%S\" is identical to \"%S\"\n", wcs1, wcs2);
    else if ( result < 0)
        printf("\"%S\" is less than \"%S\"\n", wcs1, wcs2);
    else
        printf("\"%S\" is greater than \"%S\"\n", wcs1, wcs2);
}</pre>
```

Related Information

- "strcoll() Compare Strings" on page 329
- "setlocale() Set Locale" on page 308
- "<wchar.h>" on page 18

wcscpy() — Copy Wide-Character Strings

Format

```
#include <wcstr.h>
wchar_t *wcscpy(wchar_t *string1, const wchar_t *string2);
```

Language Level: XPG4

Description

The wcscpy() function copies the contents of *string2* (including the ending wchar_t null character) into *string1*.

The wcscpy() function operates on null-ended wchar_t strings; string arguments to this function should contain a wchar t null character marking the end of the string. Only string2 needs to contain a null character. Boundary checking is not performed.

Return Value

The wcscpy() function returns a pointer to *string1*.

Example that uses wcscpy()

This example copies the contents of source to destination.

```
#include <stdio.h>
#include <wcstr.h>
#define SIZE
int main(void)
 wchar t source[ SIZE ] = L"This is the source string";
 wchar_t destination[ SIZE ] = L"And this is the destination string";
 wchar_t * return_string;
 printf( "destination is originally = \"%1s\"\n", destination );
 return_string = wcscpy( destination, source );
 printf( "After wcscpy, destination becomes \"%1s\"\n", destination );
/****** Output should be similar to: **********
destination is originally = "And this is the destination string"
After wcscpy, destination becomes "This is the source string"
*/
```

Related Information

- "strcpy() Copy Strings" on page 330
- "strncpy() Copy Strings" on page 346
- "wcscat() Concatenate Wide-Character Strings" on page 400
- "wcschr() Search for Wide Character" on page 402
- "wcscmp() Compare Wide-Character Strings" on page 403
- "wcscspn() Find Offset of First Wide-Character Match"
- "wcslen() Calculate Length of Wide-Character String" on page 409
- "wcsncpy() Copy Wide-Character Strings" on page 413
- "<wcstr.h>" on page 18

wcscspn() — Find Offset of First Wide-Character Match

Format

```
#include <wcstr.h>
size_t wcscspn(const wchar_t *string1, const wchar_t *string2);
```

Language Level: XPG4

Description

The wcscspn() function determines the number of wchar_t characters in the initial segment of the string pointed to by *string1* that do not appear in the string pointed to by *string2*.

The wcscspn() function operates on null-ended wchar_t strings; string arguments to this function should contain a wchar_t null character marking the end of the string.

Return Value

The wcscspn() function returns the number of wchar_t characters in the segment.

Example that uses wcscspn()

This example uses wcscspn() to find the first occurrence of any of the characters a, x, l, or e in string.

Related Information

- "strcspn() Find Offset of First Character Match" on page 331
- "strspn() —Find Offset of First Non-matching Character" on page 355
- "wcscat() Concatenate Wide-Character Strings" on page 400
- "wcschr() Search for Wide Character" on page 402
- "wcscmp() Compare Wide-Character Strings" on page 403
- "wcscpy() Copy Wide-Character Strings" on page 405
- "wcslen() Calculate Length of Wide-Character String" on page 409
- "wcsspn() Find Offset of First Non-matching Wide Character" on page 420
- "wcswcs() Locate Wide-Character Substring" on page 432
- "<wcstr.h>" on page 18

wcsftime() — Convert to Formatted Date and Time

Format

Language Level: ANSI

Description

The wcsftime() function converts the time and date specification in the timeptr structure into a wide-character string. It then stores the null-ended string in the array pointed to by wdest according to the format string pointed to by format. The maxsize value specifies the maximum number of wide characters that can be copied into the array. This function is equivalent to strftime(), except that it uses wide characters.

The wcsftime() function works just like the strftime() function, except that it uses wide characters. The format string is a wide-character character string that contains:

- Conversion-specification characters.
- Ordinary wide characters, which are copied into the array unchanged.

Note: When wcsftime() is compiled with LOCALETYPE(*LOCALEUCS2) wdest and format, both must be UNICODE-encoded strings, as opposed to wide EBCDIC. This function is availabe only when LOCALETYPE is (*LOCALE) or (*LOCALEUCS2).

This function uses the time structure pointed to by timeptr, and if the specifier is locale sensitive, then it will also use the LC_TIME category of the current locale to determine the appropriate replacement value of each valid specifier. The time structure pointed to by timeptr is usually obtained by calling the qmtime() or localtime() function.

Return Value

If the total number of wide characters in the resulting string, including the ending null wide character, does not exceed maxsize, wcsftime() returns the number of wide characters placed into wdest, not including the ending null wide character. Otherwise, the wcsftime() function returns 0 and the contents of the array are indeterminate.

Example that uses wcsftime()

This example obtains the date and time using localtime(), formats the information with the wcsftime(), and prints the date and time.

```
#include <stdio.h>
#include <time.h>
#include <wchar.h>
int main(void)
  struct tm *timeptr;
  wchar t dest[100];
  time t
            temp;
  size t
            rc;
   temp = time(NULL);
   timeptr = localtime(&temp);
   rc = wcsftime(dest, sizeof(dest), L" Today is %A,"
                 L" %b %d.\n Time: %I:%M %p", timeptr);
   printf("%d characters placed in string to make:\n\n%ls\n", rc, dest);
   return 0;
```

- "gmtime() Convert Time" on page 141
- "localtime() Convert Time" on page 163
- "strftime() Convert Date/Time to String" on page 336
- "strptime()— Convert String to Date/Time" on page 350
- "<wchar.h>" on page 18

wcslen() — Calculate Length of Wide-Character String

Format

```
#include <wcstr.h>
size_t wcslen(const wchar_t *string);
```

Language Level: XPG4

Description

The wcslen() function computes the number of wide characters in the string pointed to by *string*.

Note: This function is available only when LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

Return Value

The wcslen() function returns the number of wide characters in *string*, excluding the ending wchar_t null character.

Example that uses wcslen()

This example computes the length of the wide-character string string.

- "mblen() Determine Length of a Multibyte Character" on page 172
- "strlen() Determine String Length" on page 341
- "wcsncat() Concatenate Wide-Character Strings"
- "wcsncmp() Compare Wide-Character Strings" on page 411
- "wcsncpy() Copy Wide-Character Strings" on page 413
- "<wcstr.h>" on page 18

wcsncat() — Concatenate Wide-Character Strings

Format

```
#include <wcstr.h>
wchar t *wcsncat(wchar t *string1, const wchar t *string2, size t count);
```

Language Level: XPG4

Description

The wcsncat() function appends up to count wide characters from string2 to the end of *string1*, and appends a wchar_t null character to the result.

The wcsncat() function operates on null-ending wide-character strings; string arguments to this function should contain a wchar_t null character marking the end of the string.

Return Value

The wcsncat() function returns *string1*.

Example that uses wcsncat()

This example demonstrates the difference between the wcscat() and wcsncat() functions. The wcscat() function appends the entire second string to the first; the wcsncat() function appends only the specified number of characters in the second string to the first.

```
#include <stdio.h>
#include <wcstr.h>
#include <string.h>
#define SIZE 40
int main(void)
  wchar t buffer1[SIZE] = L"computer";
  wchar_t * ptr;
  /* Call wcscat with buffer1 and " program" */
  ptr = wcscat( buffer1, L" program" );
  printf( "wcscat : buffer1 = \"%ls\"\n", buffer1 );
  /* Reset buffer1 to contain just the string "computer" again */
 memset( buffer1, L'\0', sizeof( buffer1 ));
ptr = wcscpy( buffer1, L"computer" );
  /* Call wcsncat with buffer1 and " program" */
  ptr = wcsncat( buffer1, L" program", 3 );
  printf( "wcsncat: buffer1 = \"%ls\"\n", buffer1 );
/****** Output should be similar to: *********
wcscat : buffer1 = "computer program"
wcsncat: buffer1 = "computer pr"
```

- "strcat() Concatenate Strings" on page 324
- "strncat() Concatenate Strings" on page 343
- "wcscat() Concatenate Wide-Character Strings" on page 400
- "wcsncmp() Compare Wide-Character Strings"
- "wcsncpy() Copy Wide-Character Strings" on page 413
- "<wcstr.h>" on page 18

wcsncmp() — Compare Wide-Character Strings

Format

```
#include <wcstr.h>
int wcsncmp(const wchar t *string1, const wchar t *string2, size t count);
```

Language Level: XPG4

Description

The wcsncmp() function compares up to *count* wide characters in *string1* to *string2*.

The wcsncmp() function operates on null-ended wide-character strings; string arguments to this function should contain a wchar_t null character marking the end of the string.

Note: This function is available only when LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

Return Value

The wcsncmp() function returns a value indicating the relationship between the two strings, as follows:

Value Meaning

Less than 0

string1 less than string2

string1 identical to string2

Greater than 0

string1 greater than string2.

Example that uses wcsncmp()

This example demonstrates the difference between the wcscmp() function, which compares the entire strings, and the wcsncmp() function, which compares only a specified number of wide characters in the strings.

```
#include <stdio.h>
#include <wcstr.h>
#define SIZE 10
int main(void)
 int result;
 int index = 3;
 wchar t buffer1[SIZE] = L"abcdefg";
 wchar t buffer2[SIZE] = L"abcfg";
 void print result( int, wchar t *, wchar t * );
 result = wcscmp( buffer1, buffer2 );
 printf( "Comparison of each character\n" );
printf( " wcscmp: " );
print_result( result, buffer1, buffer2 );
 result = wcsncmp( buffer1, buffer2, index);
 printf( "\nComparison of only the first %i characters\n", index );
 printf( " wcsncmp: " );
 print_result( result, buffer1, buffer2 );
void print result( int res, wchar t * p buffer1, wchar t * p buffer2 )
 if ( res == 0 )
    printf( "\"%ls\" is identical to \"%ls\"\n", p buffer1, p buffer2);
 else if (res < 0)
    printf( "\"%ls\" is less than \"%ls\"\n", p_buffer1, p_buffer2 );
    printf( "\"%ls\" is greater than \"%ls\"\n", p_buffer1, p_buffer2 );
/****** Output should be similar to: *********
Comparison of each character
 wcscmp: "abcdefg" is less than "abcfg"
Comparison of only the first 3 characters
 wcsncmp: "abcdefg" is identical to "abcfg"
```

- "strcmp() Compare Strings" on page 326
- "strcoll() Compare Strings" on page 329

- "strncmp() Compare Strings" on page 344
- "wcscmp() Compare Wide-Character Strings" on page 403
- "wcsncat() Concatenate Wide-Character Strings" on page 410
- "wcsncpy() Copy Wide-Character Strings"
- "<wcstr.h>" on page 18

wcsncpy() — Copy Wide-Character Strings

Format

```
#include <wcstr.h>
wchar t *wcsncpy(wchar t *string1, const wchar t *string2, size t count);
```

Language Level: XPG4

Description

The wcsncpy() function copies up to *count* wide characters from *string2* to *string1*. If *string2* is shorter than *count* characters, *string1* is padded out to *count* characters with wchar_t null characters.

The wcsncpy() function operates on null-ended wide-character strings; string arguments to this function should contain a wchar_t null character marking the end of the string. Only *string2* needs to contain a null character.

Return Value

The wcsncpy() returns a pointer to *string1*.

Related Information

- "strcpy() Copy Strings" on page 330
- "strncpy() Copy Strings" on page 346
- "wcscpy() Copy Wide-Character Strings" on page 405
- "wcsncat() Concatenate Wide-Character Strings" on page 410
- "wcsncmp() Compare Wide-Character Strings" on page 411
- "<wcstr.h>" on page 18

_wcsicmp — Compare Wide Character Strings without Case Sensitivity

```
Format

#include <wchar.h>
int __wcsicmp(const wchar_t *string1, const wchar_t *string2);

Language Level: Extension

Thread Safe: YES.

Description

The __wcsicmp() function compares string1 and string2 without sensitivity to case.
All alphabetic wide characters in string1 and string2 are converted to lowercase before comparison. The function operates on null terminated wide character
```

strings. The string arguments to the function are expected to contain a wchar_t null character (L'\0') marking the end of the string.

Return Value

The wcsicmp() function returns a value indicating the relationship between the two strings as follows:

Table 10. Return values of wcsicmp()

Value	Meaning
Less than 0	string1 less than string2
0	string1 equivalent to string2
Greater than 0	string1 greater than string2

Example that uses __wcsicmp()

```
This example uses _wcsicmp() to compare two wide character strings.
#include <stdio.h>
#include <wchar.h>
int main(void)
 wchar t *str1 = L"STRING";
 wchar_t *str2 = L"string";
 int result;
 result = wcsicmp(str1, str2);
 if (result == 0)
   printf("Strings compared equal.\n");
 else if (result < 0)
   printf("\"%ls\" is less than \"%ls\".\n", str1, str2);
   printf("\"%ls\" is greater than \"%ls\".\n", str1, str2);
 return 0;
/***** The output should be similar to: ********
Strings compared equal.
****************************
```

- "strcmp() Compare Strings" on page 326
- "strncmp() Compare Strings" on page 344
- "wcscat() Concatenate Wide-Character Strings" on page 400
- "wcschr() Search for Wide Character" on page 402
- "wcscspn() Find Offset of First Wide-Character Match" on page 406
- "wcslen() Calculate Length of Wide-Character String" on page 409
- "wcsncmp() Compare Wide-Character Strings" on page 411
- "_wcsnicmp Compare Wide Character Strings without Case Sensitivity" on page 415
- "<wchar.h>" on page 18

__wcsnicmp — Compare Wide Character Strings without Case Sensitivity

Format

```
#include <wchar.h>;
int __wcsnicmp(const wchar_t *string1, const wchar_t *string2, size_t count);
```

Language Level: Extension

Thread Safe: YES.

Description

The _wcsnicmp() function compares up to count characters of *string1* and *string2* without sensitivity to case. All alphabetic wide characters in *string1* and *string2* are converted to lowercase before comparison.

The _wcsnicmp() function operates on null terminated wide character strings. The string arguments to the function are expected to contain a wchar_t null character $(L'\setminus 0')$ marking the end of the string.

Return Value

The_wcsnicmp() function returns a value indicating the relationship between the two strings, as follows:

Table 11. Return values of _wcsicmp()

Value	Meaning
Less than 0	string1 less than string2
0	string1 equivalent to string2
Greater than 0	string1 greater than string2

.

Example that uses __wcsnicmp()

Strings compared equal. ************

Related Information

- "strcmp() Compare Strings" on page 326
- "strncmp() Compare Strings" on page 344
- "wcscat() Concatenate Wide-Character Strings" on page 400
- "wcschr() Search for Wide Character" on page 402
- "wcscspn() Find Offset of First Wide-Character Match" on page 406
- "wcslen() Calculate Length of Wide-Character String" on page 409
- "wcsncmp() Compare Wide-Character Strings" on page 411
- "_wcsicmp Compare Wide Character Strings without Case Sensitivity" on page 413
- "<wchar.h>" on page 18

wcspbrk() — Locate Wide Characters in String

Format

```
#include <wcstr.h>
wchar t *wcspbrk(const wchar t *string1, const wchar t *string2);
```

Language Level: XPG4

Description

The wcspbrk() function locates the first occurrence in the string pointed to by string1 of any wide character from the string pointed to by string2.

Return Value

The wcspbrk() function returns a pointer to the character. If string1 and string2 have no wide characters in common, the wcspbrk() function returns NULL.

Example that uses wcspbrk()

This example uses wcspbrk() to find the first occurrence of either "a" or "b" in the array string.

```
#include <stdio.h>
#include <wcstr.h>
int main(void)
 wchar t * result;
 wchar_t * string = L"The Blue Danube";
 wchar t *chars = L"ab";
 result = wcspbrk( string, chars);
  printf("The first occurrence of any of the characters \"%1s\" in "
        "\"%ls\" is \"%ls\"\n", chars, string, result);
/****** Output should be similar to: *********
The first occurrence of any of the characters "ab" in "The Blue Danube"
```

- "strchr() Search for Character" on page 325
- "strcspn() Find Offset of First Character Match" on page 331
- "strpbrk() Find Characters in String" on page 348
- "strspn() —Find Offset of First Non-matching Character" on page 355
- "wcschr() Search for Wide Character" on page 402
- "wcscmp() Compare Wide-Character Strings" on page 403
- "wcscspn() Find Offset of First Wide-Character Match" on page 406
- "wcsncmp() Compare Wide-Character Strings" on page 411
- "wcsrchr() Locate Last Occurrence of Wide Character in String"
- "wcswcs() Locate Wide-Character Substring" on page 432
- "<wcstr.h>" on page 18

wcsrchr() — Locate Last Occurrence of Wide Character in String

Format

```
#include <wcstr.h>
wchar t *wcsrchr(const wchar t *string, wchar t character);
```

Description

The wcsrchr() function locates the last occurrence of *character* in the string pointed to by string. The ending wchar_t null character is considered to be part of the string.

Return Value

The wcsrchr() function returns a pointer to the character, or a NULL pointer if character does not occur in the string.

Example that uses wcsrchr()

This example compares the use of wcschr() and wcsrchr(). It searches the string for the first and last occurrence of p in the wide character string.

```
#include <stdio.h>
#include <wcstr.h>
#define SIZE 40
int main(void)
 wchar t buf[SIZE] = L"computer program";
 wchar_t * ptr;
 int ch = 'p';
 /* This illustrates wcschr */
 ptr = wcschr( buf, ch );
 printf( "The first occurrence of %c in '%1s' is '%1s'\n", ch, buf, ptr );
 /* This illustrates wscrchr */
 ptr = wcsrchr( buf, ch );
 printf( "The last occurrence of %c in '%ls' is '%ls'\n", ch, buf, ptr );
/******* Output should be similar to: **********
The first occurrence of p in 'computer program' is 'puter program'
The last occurrence of p in 'computer program' is 'program'
```

- "strchr() Search for Character" on page 325
- "strrchr() Locate Last Occurrence of Character in String" on page 354
- "strcspn() Find Offset of First Character Match" on page 331
- "strspn() —Find Offset of First Non-matching Character" on page 355
- "wcschr() Search for Wide Character" on page 402
- "wcscmp() Compare Wide-Character Strings" on page 403
- "wcscspn() Find Offset of First Wide-Character Match" on page 406
- "wcsncmp() Compare Wide-Character Strings" on page 411
- "wcswcs() Locate Wide-Character Substring" on page 432
- "wcspbrk() Locate Wide Characters in String" on page 416
- "<wcstr.h>" on page 18

wcsrtombs() — Convert Wide Character String to Multibyte String (Restartable)

Format

```
#include <wchar.h>
size_t wcsrtombs (char *dst, const wchar_t **src, size_t len,
                 mbstate t *ps);
```

Language Level: ANSI

Thread Safe: Yes, if the fourth parameter, ps, is not NULL.

Description

This function is the restartable version of wcstombs().

1

The wcsrtombs() function converts a sequence of wide characters from the array indirectly pointed to by src into a sequence of corresponding multibyte characters that begins in the shift state described by ps, which, if dst is not a null pointer, are then stored into the array pointed to by dst. Conversion continues up to and including the ending null wide character, but the ending null character (byte) will not be stored. Conversion will stop earlier in two cases: when a code is reached that does not correspond to a valid multibyte character, or (if dst is not a null pointer) when the next multibyte element would exceed the limit of len total bytes to be stored into the array pointed to by dst. Each conversion takes place as if by a call to wcrtomb().

If dst is not a null pointer, the object pointed to by src will be assigned either a null pointer (if conversion stopped due to reaching a ending null character) or the address of the code just past the last wide character converted. If conversion stopped due to reaching a ending null wide character, the resulting state described will be the initial conversion state.

Note: This function is accessible only if LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

If the program is compiled with LOCALETYPE(*LOCALEUCS2), it will behave as if it was compiled with LOCALETYPE(*LOCALE), since this function does not support UNICODE yet.

Return Value

If the first code is not a valid wide character, an encoding error will occur. wcsrtombs() stores the value of the macro EILSEQ in errno and returns (size t) -1, but the conversion state will be unchanged. Otherwise it returns the number of bytes in the resulting multibyte character sequence, which is the same as the number of array elements changed when dst is not a null pointer.

Example that uses wcsrtombs()

```
#include <stdio.h>
#include <wchar.h>
#include <string.h>
#define SIZE 20
int main(void)
  char
           dest[SIZE];
  wchar_t *wcs = L"string";
  wchar t *ptr;
  size_t count = SIZE;
size_t length;
  mbstate_t ps = 0;
  ptr = (wchar t *) wcs;
  length = wcsrtombs(dest, ptr, count, &ps);
  printf("%d characters were converted.\n", length);
  printf("The converted string is \"%s\"\n\n", dest);
   /* Reset the destination buffer */
  memset(dest, '\0', sizeof(dest));
  /* Now convert only 3 characters */
  ptr = (wchar t *) wcs;
   length = wcsrtombs(dest, ptr, 3, &ps);
  printf("%d characters were converted.\n", length);
  printf("The converted string is \"%s\"\n\n", dest);
/****** Output should be similar to: ***********
6 characters were converted.
The converted string is "string"
3 characters were converted.
The converted string is "str"
```

Related Information

- "mblen() Determine Length of a Multibyte Character" on page 172
- "mbrlen() Determine Length of a Multibyte Character (Restartable)" on page 173
- "mbrtowc() Convert a Multibyte Character to a Wide Character (Restartable)" on page 176
- "mbsrtowcs() Convert a Multibyte String to a Wide Character String (Restartable)" on page 180
- "wcrtomb() Convert a Wide Character to a Multibyte Character (Restartable)" on page 396
- "wcstombs() Convert Wide-Character String to Multibyte String" on page 426
- "<wchar.h>" on page 18

wcsspn() — Find Offset of First Non-matching Wide Character

Format

```
#include <wcstr.h>
size_t wcsspn(const wchar_t *string1, const wchar_t *string2);
```

Description

The wcsspn() function computes the number of wide characters in the initial segment of the string pointed to by *string1*, which consists entirely of wide characters from the string pointed to by *string2*.

Return Value

The wcsspn() function returns the number of wide characters in the segment.

Example that uses wcsspn()

This example finds the first occurrence in the array string of a wide character that is not an a, b, or c. Because the string in this example is cabbage, the wcsspn() function returns 5, the index of the segment of cabbage before a character that is not an a, b, or c.

Related Information

- "strchr() Search for Character" on page 325
- "strcspn() Find Offset of First Character Match" on page 331
- "strpbrk() Find Characters in String" on page 348
- "strrchr() Locate Last Occurrence of Character in String" on page 354
- "strspn() —Find Offset of First Non-matching Character" on page 355
- "wcscat() Concatenate Wide-Character Strings" on page 400
- "wcschr() Search for Wide Character" on page 402
- "wcscmp() Compare Wide-Character Strings" on page 403
- "wcscspn() Find Offset of First Wide-Character Match" on page 406
- "wcsncmp() Compare Wide-Character Strings" on page 411
- "wcspbrk() Locate Wide Characters in String" on page 416
- "wcsrchr() Locate Last Occurrence of Wide Character in String" on page 417
- "wcsspn() Find Offset of First Non-matching Wide Character" on page 420
- "wcswcs() Locate Wide-Character Substring" on page 432
- "<wcstr.h>" on page 18

wcsstr() — Locate Wide-Character Substring

Format

```
#include <wchar.h>
wchar t *wcsstr(const wchar t *wcs1, const wchar t *wcs2);
```

Language Level: ANSI

Description

The wcsstr() function locates the first occurrence of wcs2 in wcs1.

Return Value

The wcsstr() function returns a pointer to the beginning of the first occurrence of wcs2 in wcs1. If wcs2 does not appear in wcs1, the wcsstr() function returns NULL. If wcs2 points to a wide-character string with zero length, it returns wcs1.

Example that uses wcsstr()

This example uses the wcsstr() function to find the first occurrence of "hay" in the wide-character string "needle in a haystack".

```
#include <stdio.h>
#include <wchar.h>
int main(void)
  wchar t *wcs1 = L"needle in a haystack";
  wchar_t *wcs2 = L"hay";
  printf("result: \"%ls\"\n", wcsstr(wcs1, wcs2));
  return 0;
     The output should be similar to:
     result: "haystack"
  ************************************
```

Related Information

- "strstr() Locate Substring" on page 356
- "wcschr() Search for Wide Character" on page 402
- "wcsrchr() Locate Last Occurrence of Wide Character in String" on page 417
- "wcswcs() Locate Wide-Character Substring" on page 432
- "<wchar.h>" on page 18

wcstod() — Convert Wide-Character String to Double

Format

```
#include <wchar.h>
double wcstod(const wchar t *nptr, wchar t **endptr);
```

Language Level: XPG4

Description

The wcstod() function converts the initial portion of the wide-character string pointed to by nptr to a double value. The nptr parameter points to a sequence of characters that can be interpreted as a numerical value of type double. The

wcstod() function stops reading the string at the first character that it cannot recognize as part of a number. This character can be the wchar_t null character at the end of the string.

The behavior of the wcstod() function is affected by the LC_CTYPE category of the current locale. This function is available only when LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

If the program is compiled LOCALETYPE(*LOCALEUCS2), then the wide characters being converted are assumed to be UNICODE characters.

Return Value

The wcstod() function returns the converted double value. If no conversion could be performed, the wcstod() function returns 0. If the correct value is outside the range of representable values, the wcstod() function returns +HUGE_VAL or -HUGE_VAL (according to the sign of the value), and sets errno to ERANGE. If the correct value would cause underflow, the wcstod() function returns 0 and sets errno to ERANGE. If the string *nptr* points to is empty or does not have the expected form, no conversion is performed, and the value of *nptr* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

The value of errno may be set to **ERANGE**, range error.

Example that uses wcstod()

This example uses the wcstod() function to convert the string wcs to a floating-point value.

```
#include <stdio.h>
#include <wchar.h>
int main(void)
  wchar t *wcs = L"3.1415926This stopped it";
  wchar t *stopwcs;
  printf("wcs = \"%ls\"\n", wcs);
  printf(" wcstod = %f\n", wcstod(wcs, &stopwcs));
  printf("
            Stop scanning at \"%ls\"\n", stopwcs);
  return 0;
  /***************
     The output should be similar to:
     wcs = "3.1415926This stopped it"
       wcstod = 3.141593
       Stop scanning at "This stopped it"
  ******************************
}
```

Related Information

- "strtod() Convert Character String to Double" on page 357
- "strtol() strtoll() Convert Character String to Long and Long Long Integer" on page 362
- "wcstol() wcstoll() Convert Wide Character String to Long and Long Long Integer" on page 424
- "wcstoul() wcstoull() Convert WideCharacter String to Unsigned Long and Unsigned Long Integer" on page 430

wcstol() — wcstoll() — Convert Wide Character String to Long and Long Long Integer

```
Format (wcstol())
#include <wchar.h>
long int wcstol(const wchar t *nptr, wchar t **endptr, int base);
Format (wcstoll())
#include <wchar.h>
long long int wcstoll(const wchar_t *nptr, wchar_t **endptr, int base);
```

Language Level: ANSI

Description

The wcstol () function converts the initial portion of the wide-character string pointed to by *nptr* to a long integer value. The *nptr* parameter points to a sequence of wide characters that can be interpreted as a numerical value of type long int. The wcstol () function stops reading the string at the first wide character that it cannot recognize as part of a number. This character can be the wchar t null character at the end of the string. The ending character can also be the first numeric character greater than or equal to the base.

The behavior of the wcstol () function is affected by the LC_CTYPE category of the current locale. This function is available only when LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

If a program is compiled with LOCALETYPE(*LOCALEUCS2), the wide characters being converted are assumed to be UNICODE characters.

The wcstoll() subroutine converts a wide-character string to a long long integer. The wide-character string is parsed to skip the initial space characters (as determined by the iswspace subroutine). Any non-space character signifies the start of a subject string that may form a long long int in the radix specified by the base parameter. The subject sequence is defined to be the longest initial substring that is a long long int of the expected form.

If the value of the *endptr* parameter is not null, then a pointer to the character that ended the scan is stored in endptr. If a long long integer cannot be formed, the value of the *endptr* parameter is set to that of the *nptr* parameter.

If the base parameter is a value between 2 and 36, the subject sequence's expected form is a sequence of letters and digits representing a long long integer whose radix is specified by the base parameter. This sequence optionally is preceded by a positive (+) or negative (-) sign. Letters from a (or A) to z (or Z) inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of the base parameter are permitted. If the base parameter has a value of 16, the characters 0x or 0X optionally precede the sequence of letters and digits, following the positive (+) or negative (-) sign, if present.

If the value of the base parameter is 0, the string determines the base. Therefore, after an optional leading sign, a leading 0 indicates octal conversion, and a leading 0x or 0X indicates hexadecimal conversion.

Return Value

1

The wcstol() function returns the converted long integer value. If no conversion could be performed, the wcstol() function returns 0. If the correct value is outside the range of representable values, the wcstol()function returns LONG_MAX or LONG_MIN (according to the sign of the value), and sets errno to ERANGE. If the string *nptr* points to is empty or does not have the expected form, no conversion is performed, and the value of *nptr* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

Upon successful completion, the wcstoll() subroutine returns the converted value. If no conversion could be performed, 0 is returned, and the errno global variable is set to indicate the error. If the correct value is outside the range of representable values, the wcstoll() subroutine returns a value of LONG_LONG_MAX or LONG_LONG_MIN.

The value of errno may be set to **ERANGE** (range error), or **EINVAL** (invalid argument).

Example that uses wcstol()

This example uses the wcstol() function to convert the wide-character string wcs to a long integer value.

```
#include <stdio.h>
#include <wchar.h>
int main(void)
  wchar t *wcs = L"10110134932";
  wchar_t *stopwcs;
  long
          1;
  int
          base:
  printf("wcs = \"%ls\"\n", wcs);
  for (base=2; base<=8; base*=2) {
     1 = wcstol(wcs, &stopwcs, base);
     printf(" wcstol = %ld\n"
              Stopped scan at \"%1s\"\n\n", 1, stopwcs);
  return 0;
  /*****************
     The output should be similar to:
     wcs = "10110134932"
       wcstol = 45
       Stopped scan at "34932"
        wcstol = 4423
        Stopped scan at "4932"
       wcstol = 2134108
       Stopped scan at "932"
  ***********************************
}
```

Related Information

 "strtol() — strtoll() — Convert Character String to Long and Long Long Integer" on page 362

- "strtoul() strtoull() Convert Character String to Unsigned Long and Unsigned Long Long Integer" on page 364
- "wcstod() Convert Wide-Character String to Double" on page 422
- "wcstoul() wcstoull() Convert WideCharacter String to Unsigned Long and Unsigned Long Long Integer" on page 430
- "<wchar.h>" on page 18

wcstombs() — Convert Wide-Character String to Multibyte String

Format

#include <stdlib.h> size t wcstombs(char *dest, const wchar t *string, size t count);

Language Level: ANSI

Thread Safe: YES.

Description

The wcstombs() function converts the wide-character string pointed to by string into the multibyte array pointed to by dest. The converted string begins in the initial shift state. The conversion stops after count bytes in dest are filled up or a wchar t null character is encountered.

Only complete multibyte characters are stored in *dest*. If the lack of space in *dest* would cause a partial multibyte character to be stored, wcstombs() stores fewer than n bytes and discards the invalid character.

The behavior of this function is affected by the LC CTYPE category of the current locale.

When the program is compiled with LOCALETYPE(*LOCALE) and SYSIFCOPT(*IFSIO), the behavior of wcstombs() is affected by the LC_CTYPE category of the current locale. Remember that the CCSID of the locale should match the CCSID of your job. If the CCSID of the locale is a single-byte CCSID, the wide characters are converted to single-byte characters. Any wide character whose value is greater than 256 would be invalid when converting to single-byte CCSID. When the CCSID is multibyte CCSID, the wide characters are converted to multibyte characters.

When the program is compiled with LOCALETYPE(*LOCALEUCS2) and SYSIFCOPT(*IFSIO), the wide characters are assumed to be UNICODE characters, and these UNICODE characters are converted to the CCSID of the locale that is associated with LC_TYPE.

Note: This function is accessible only if LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

Return Value

The wcstombs() function returns the length in bytes of the multibyte character string, not including a ending null character. The value (size_t)-1 is returned if an invalid multibyte character is encountered.

The value of errno may be set to EILSER (conversion stopped due to input character), or ECONVERT (conversion error).

Examples that use wcstombs()

string = ABC, length = 3

> This program is compiled with LOCALETYPE(*LOCALE) and SYSIFCOPT(*IFSIO): #include <stdio.h> #include <stdlib.h> #include <locale.h> #include <wchar.h> #define STRLENGTH 10 "qsys.lib/JA JP.locale" #define LOCNAME #define LOCNAME EN "qsys.lib/EN US.locale" int main(void) string[STRLENGTH]; char int length, s1 = 0; wchar_t wc2[] = L"ABC"; wchar t wc string[10]; $mbsta\overline{t}e_t ps = 0;$ memset(string, '\0', STRLENGTH); $wc_string[0] = 0x00C1;$ wc string[1] = 0x4171; $wc_string[2] = 0x4172;$ wc string[3] = 0x00C2;wc string[4] = 0x0000; /* In this first example we will convert a wide character string */ /* to a single byte character string. We first set the locale */ /* to a single byte locale. We choose a locale with /* CCSID 37. */ if (setlocale(LC_ALL, LOCNAME_EN) == NULL) printf("setlocale failed. $\n"$); length = wcstombs(string, wc2, 10); /* In this case wide characters ABC are converted to */ /* single byte characters ABC, length is 3. printf("string = %s, length = %d\n\n", string, length); /* Now lets try a multibyte example. We first must set the */ /* locale to a multibyte locale. We choose a locale with /* CCSID 5026 */ if (setlocale(LC ALL, LOCNAME) == NULL) printf("setlocale failed.\n"); length = wcstombs(string, wc string, 10); /* The hex look at string would now be: /* C10E417141720FC2 length will be 8 */ /* You would need a device capable of displaying multibyte */ /* characters to see this string. printf("length = %d\n\n", length); /* The output should look like this:

```
This program is compiled with LOCALETYPE(*LOCALEUCS2) and
SYSIFCOPT(*IFSIO):
```

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <wchar.h>
#define STRLENGTH
#define LOCNAME
                     "gsys.lib/JA JP.locale"
#define LOCNAME EN "qsys.lib/EN US.locale"
int main(void)
             string[STRLENGTH];
    char
   int length, s1 = 0;
wchar_t wc2[] = L"ABC";
   wchar_t wc_string[10];
   mbstate t ps = 0;
   memset(string, '\0', STRLENGTH);
    wc_string[0] = 0x0041;
                                /* UNICODE A */
    wc_string[1] = 0xFF41;
   wc string[2] = 0xFF42;
   wc string[3] = 0x0042;
                                 /* UNICODE B */
   wc string[4] = 0x0000;
    /* In this first example we will convert a wide character string */
   /* to a single byte character string. We first set the locale */
    /* to a single byte locale. We choose a locale with
                                                                     */
    /* CCSID 37.
                                                                      */
   if (setlocale(LC_ALL, LOCNAME_EN) == NULL)
       printf("setlocale failed.\sqrt{n}");
    length = wcstombs(string, wc2, 10);
    /* In this case wide characters ABC are converted to */
    /* single byte characters ABC, length is 3. */
    printf("string = %s, length = %d\n\n", string, length);
    /* Now lets try a multibyte example. We first must set the */
    /* locale to a multibyte locale. We choose a locale with
    /* CCSID 5026 */
    if (setlocale(LC ALL, LOCNAME) == NULL)
       printf("setlocale failed.\n");
    length = wcstombs(string, wc_string, 10);
    /* The hex look at string would now be:
    /* C10E428142820FC2 length will be 8
   /* You would need a device capable of displaying multibyte */
    /* characters to see this string.
    printf("length = %d\n\n", length);
/* The output should look like this:
string = ABC, length = 3
```

*/

Related Information

- "mbstowcs() Convert a Multibyte String to a Wide Character String" on page 182
- "wcslen() Calculate Length of Wide-Character String" on page 409
- "wcsrtombs() Convert Wide Character String to Multibyte String (Restartable)" on page 418
- "wctomb() Convert Wide Character to Multibyte Character" on page 436
- "<stdlib.h>" on page 16

wcstok() — Tokenize Wide-Character String

Format

```
#include <wchar.h>
wchar t *wcstok(wchar t *wcs1, const wchar t *wcs2, wchar t **ptr);
```

Language Level: ANSI

Description

The wcstok() function reads wcs1 as a series of zero or more tokens and wcs2 as the set of wide characters serving as delimiters for the tokens in wcs1. A sequence of calls to the wcstok() function locates the tokens inside wcs1. The tokens can be separated by one or more of the delimiters from wcs2. The third argument points to a wide-character pointer that you provide where the wcstok() function stores information necessary for it to continue scanning the same string.

When the wcstok() function is first called for the wide-character string wcs1, it searches for the first token in wcs1, skipping over leading delimiters. The wcstok() function returns a pointer to the first token. To read the next token from wcs1, call the wcstok() function with NULL as the first parameter (wcs1). This NULL parameter causes the wcstok() function to search for the next token in the previous token string. Each delimiter is replaced by a null character to end the token.

The wcstok() function always stores enough information in the pointer *ptr* so that subsequent calls, with NULL as the first parameter and the unmodified pointer value as the third, will start searching right after the previously returned token. You can change the set of delimiters (*wcs2*) from call to call.

Return Value

The wcstok() function returns a pointer to the first wide character of the token, or a null pointer if there is no token. In later calls with the same token string, the wcstok() function returns a pointer to the next token in the string. When there are no more tokens, the wcstok() function returns NULL.

Example that uses wcstok()

This example uses the wcstok() function to locate the tokens in the wide-character string str1.

```
#include <stdio.h>
#include <wchar.h>
int main(void)
  static wchar t str1[] = L"?a??b,,,#c";
  static wchar_t str2[] = L"\t \t";
  wchar_t *t, *ptr1, *ptr2;
  t = wcstok(str1, L"?", &ptr1); /* t points to the token L"a" */
  printf("t = '%1s'\n", t);
t = wcstok(NULL, L",", &printf("t = '%1s'\n", t);
                     , &ptrl); /* t points to the token L"?b"*/
  t = wcstok(str2, L" \t,", &ptr2); /* t is a null pointer
  printf("t = '%1s'\n", t);
  t = wcstok(NULL, L"#,", &ptr1); /* t points to the token L"c" */
  printf("t = '%ls'\n", t);
  t = wcstok(NULL, L"?", &ptr1);  /* t is a null pointer
                                                                */
  printf("t = \frac{1}{3} \ln \frac{1}{n}, t);
  return 0;
  The output should be similar to:
         t = 'a'
        t = '?b'
        t = ''
        t = 'c'
        t = ''
  }
```

Related Information

- "strtok() Tokenize String" on page 359
- "<wchar.h>" on page 18

wcstoul() — wcstoull() — Convert WideCharacter String to Unsigned Long and Unsigned Long Long Integer

```
Format (wcstoul())
#include <wchar.h>
unsigned long int wcstoul(const wchar_t *nptr, wchar_t **endptr, int base);
Format (wcstoull())
#include <wchar.h>
unsigned long long int wcstoull(const wchar_t *nptr, wchar_t **endptr, int base);
Language Level: ANSI
```

Description

The wcstoul () function converts the initial portion of the wide-character string pointed to by nptr to an unsigned long integer value. The nptr parameter points to a sequence of wide characters that can be interpreted as a numerical value of type unsigned long int. The wcstoul () function stops reading the string at the first wide character that it cannot recognize as part of a number. This character can be the wchar_t null character at the end of the string. The ending character can also be the first numeric character greater than or equal to the base.

The behavior of the wcstoul () function is affected by the LC_CTYPE category of the current locale. This function is available only when LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

The wcstoull() subroutine converts a wide-character string to an unsigned long long integer. The wide-character string is parsed to skip the initial space characters (as determined by the iswspace subroutine). Any non-space character signifies the start of a subject string that may form an unsigned long long int in the radix specified by the base parameter. The subject sequence is defined to be the longest initial substring that is an unsigned long long int of the expected form.

If the value of the endptr parameter is not null, then a pointer to the character that ended the scan is stored in endptr. If an unsigned long long integer cannot be formed, the value of the *endptr* parameter is set to that of the *nptr* parameter.

If the base parameter is a value between 2 and 36, the subject sequence's expected form is a sequence of letters and digits representing an unsigned long long integer whose radix is specified by the base parameter. This sequence optionally is preceded by a positive (+) or negative (-) sign. Letters from a (or A) to z (or Z) inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of the base parameter are permitted. If the base parameter has a value of 16, the characters 0x or 0X optionally precede the sequence of letters and digits, following the positive (+) or negative (-) sign, if present.

If the value of the base parameter is 0, the string determines the base. Therefore, after an optional leading sign, a leading 0 indicates octal conversion, and a leading 0x or 0X indicates hexadecimal conversion.

The value of errno may be set to EINVAL (endptr is null, no numbers are found, or base is invalid), or **ERANGE** (converted value is outside the range).

Return Value

> The wcstoul() function returns the converted unsigned long integer value. If no conversion could be performed, the wcstoul () function returns 0. If the correct value is outside the range of representable values, The wcstoul () function returns ULONG_MAX and sets errno to ERANGE. If the string *nptr* points to is empty or does not have the expected form, no conversion is performed, and the value of nptr is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

Upon successful completion, the wcstoull() subroutine returns the converted value. If no conversion could be performed, 0 is returned, and the errno global variable is set to indicate the error. If the correct value is outside the range of representable values, wcstoull() subroutine returns a value of ULONG_LONG_MAX.

Example that uses wcstoul()

This example uses the wcstoul () function to convert the string wcs to an unsigned long integer value.

```
#include <stdio.h>
#include <wchar.h>
#define BASE 2
int main(void)
```

```
wchar t *wcs = L"1000e13 camels";
  wchar t *endptr;
  unsigned long int answer;
  answer = wcstoul(wcs, &endptr, BASE);
  printf("The input wide string used: '%ls'\n"
        "The unsigned long int produced: %lu\n"
        "The substring of the input wide string that was not"
        " converted to unsigned long: '%ls'\n", wcs, answer, endptr);
  return 0;
  The output should be similar to:
    The input wide string used: 1000e13 camels
    The unsigned long int produced: 8
    The substring of the input wide string that was not converted to
    unsigned long: e13 camels
  ******************************
}
```

Related Information

- "strtod() Convert Character String to Double" on page 357
- "strtol() strtoll() Convert Character String to Long and Long Long Integer" on page 362
- "wcstod() Convert Wide-Character String to Double" on page 422
- "wcstol() wcstoll() Convert Wide Character String to Long and Long Long Integer" on page 424
- "<wchar.h>" on page 18

wcswcs() — Locate Wide-Character Substring

Format

```
#include <wcstr.h>
wchar t *wcswcs(const wchar t *string1, const wchar t *string2);
```

Language Level: XPG4

Description

The wcswcs() function locates the first occurrence of string2 in the wide-character string pointed to by string1. In the matching process, the wcswcs() function ignores the wchar_t null character that ends *string2*.

Return Value

The wcswcs () function returns a pointer to the located string or NULL if the string is not found. If *string2* points to a string with zero length, wcswcs() returns *string1*.

Example that uses wcswcs()

This example finds the first occurrence of the wide character string pr in buffer1.

Related Information

- "strchr() Search for Character" on page 325
- "strcspn() Find Offset of First Character Match" on page 331
- "strpbrk() Find Characters in String" on page 348
- "strrchr() Locate Last Occurrence of Character in String" on page 354
- "strspn() —Find Offset of First Non-matching Character" on page 355
- "strstr() Locate Substring" on page 356
- "wcschr() Search for Wide Character" on page 402
- "wcscmp() Compare Wide-Character Strings" on page 403
- "wcscspn() Find Offset of First Wide-Character Match" on page 406
- "wcspbrk() Locate Wide Characters in String" on page 416
- "wcsrchr() Locate Last Occurrence of Wide Character in String" on page 417
- "wcsspn() Find Offset of First Non-matching Wide Character" on page 420
- "<wcstr.h>" on page 18

wcswidth() — Determine the Display Width of a Wide Character String

Format

I

```
#include <wchar.h>
int wcswidth (const wchar_t *wcs, size_t n);
```

Language Level: XPG4

Thread Safe: YES.

Description

The wcswidth() function determines the number of printing positions that a graphic representation of n wide characters (or fewer that n wide characters if a null wide character is encountered before n wide characters have been exhausted) in the wide string pointed to by wcs occupies on a display device. The number is independent of its location on the device.

The value of errno may be set to **EILSEQ** (invalid wide character), or **ECONVERT** (conversion error).

Return Value

The wcswidth() function either returns:

- 0, if wcs points to a null wide character; or
- the number of printing positions occupied by the wide string pointed to by wcs;
- -1, if any wide character in the wide string pointed to by wcs is not a printing wide character.

The behavior of the wcswidth() function is affected by the LC_CTYPE category.

Example that uses wcswidth()

```
#include <stdio.h>
#include <wchar.h>
int main(void)
  wchar t *wcs = L"ABC";
  printf("wcs has a width of: d\n", wcswidth(wcs,3));
/*********The output is as follows********/
*/
              wcs has a width of: 3
                                                    */
```

Related Information

- "wcswidth() Determine the Display Width of a Wide Character String" on
- "<wchar.h>" on page 18

wcsxfrm() — Transform a Wide-Character String

Format

```
#include <wchar.h>
size_t wcsxfrm (wchar_t *wcs1, const wchar_t *wcs2, size_t n);
```

Language Level: XPG4

Description

The wcsxfrm() function transforms the wide-character string pointed to by wcs2 to values which represent character collating weights and places the resulting wide-character string into the array pointed to by wcs1.

Note: This function is accessible only if LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command. The behavior of this wide-character function is affected by the LC_COLLATE category of the current locale.

Return Value

The wcsxfrm() function returns the length of the transformed wide-character string (not including the ending null wide character code). If the value returned is n or more, the contents of the array pointed to by wcs1 are indeterminate.

If wcsxfrm() is unsuccessful, errno is changed. The value of errno may be set to EINVAL (the *wcs1* or *wcs2* arguments contain characters which are not available in the current locale).

Example that uses wcsxfrm()

```
#include <stdio.h>
#include <wchar.h>

int main(void)
{
    wchar_t *wcs;
    wchar_t buffer[80];
    int length;

    printf("Type in a string of characters.\n ");
    wcs = fgetws(buffer, 80, stdin);
    length = wcsxfrm(NULL, wcs, 0);
    printf("You would need a %d element array to hold the wide string\n", length);
    printf("\n\n%S\n\n transformed according", wcs);
    printf(" to this program's locale. \n");
}
```

Related Information

- "strxfrm() Transform String" on page 366
- "<wchar.h>" on page 18

wctob() — Convert Wide Character to Byte

Format

```
#include <stdio.h>
#include <wchar.h>
int wctob(wint_t wc);
```

Language Level: ANSI

Description

The wctob() function determines whether *wc* corresponds to a member of the extended character set, whose multibyte character has a length of 1 byte when in the initial shift state. The behavior of the wctob() function is affected by the LC_CTYPE category of the current locale. This function is available only when LOCALETYPE(*LOCALE) is specified on the compilation command.

If a program is compiled with LOCALETYPE(*LOCALEUCS2), the wctob() function will behave as if the program was compiled with LOCALETYPE(*LOCALE), because the wctob() function does not support UNICODE.

Return Value

If *c* corresponds to a multibyte character with a length of 1 byte, the wctob() function returns the single-byte representation. Otherwise, it returns EOF.

Example that uses wctob()

This example uses the wctob() function to test if the wide character A is a valid single-byte character.

```
#include <stdio.h>
#include <wchar.h>
int main(void)
  wint t wc = L'A';
  if (wctob(wc) == wc)
     printf("%lc is a valid single byte character\n", wc);
     printf("%lc is not a valid single byte character\n", wc);
  return 0;
  /***********************
     The output should be similar to:
     A is a valid single byte character
```

Related Information

- "mbtowc() Convert Multibyte Character to a Wide Character" on page 186
- "wctomb() Convert Wide Character to Multibyte Character"
- "wcstombs() Convert Wide-Character String to Multibyte String" on page 426
- "<wchar.h>" on page 18

wctomb() — Convert Wide Character to Multibyte Character

Format

```
#include <stdlib.h>
int wctomb(char *string, wchar t character);
```

Language Level: ANSI

Description

Thread Safe: NO. Use wcrtomb() instead.

The wctomb() function converts the wchar_t value of *character* into a multibyte array pointed to by string. If the value of character is 0, the function is left in the initial shift state. At most, the wctomb() function stores MB_CUR_MAX characters in string.

The conversion of the wide character is the same as described in wcstombs(). See this function for a UNICODE example.

Return Value

The wctomb() function returns the length in bytes of the multibyte character. The value -1 is returned if *character* is not a valid multibyte character. If *string* is a NULL pointer, the wctomb() function returns nonzero if shift-dependent encoding is used, or 0 otherwise.

Example that uses wctomb()

This example converts the wide character c to a multibyte character.

```
#include <stdio.h>
#include <stdlib.h>
#include <wcstr.h>
#define SIZE 40
int main(void)
 static char buffer[ SIZE ];
 wchar t wch = L'c';
 int length;
 length = wctomb( buffer, wch );
 printf( "The number of bytes that comprise the multibyte "
            "character is %i\n", length );
 printf( "And the converted string is \"%s\"\n", buffer );
/****** Output should be similar to: *********
The number of bytes that comprise the multibyte character is 1
And the converted string is "c"
*/
```

Related Information

- "mbtowc() Convert Multibyte Character to a Wide Character" on page 186
- "wcslen() Calculate Length of Wide-Character String" on page 409
- "wcrtomb() Convert a Wide Character to a Multibyte Character (Restartable)" on page 396
- "wcstombs() Convert Wide-Character String to Multibyte String" on page 426
- "wcsrtombs() Convert Wide Character String to Multibyte String (Restartable)" on page 418
- "<stdlib.h>" on page 16

wctrans() —Get Handle for Character Mapping

Format

```
#include <wctype.h>
wctrans_t wctrans(const char *property);
```

Language Level: ANSI

Description

The wctrans() function constructs a value with type wctrans_t. This value describes a mapping between wide characters identified by the string argument *property*. The two strings listed under the towctrans() function description shall be valid in all locales as *property* arguments to the wctrans() function.

Return Value

If property identifies a valid mapping of wide characters according to the LC CTYPE category of the current locale, the wctrans() function returns a nonzero value that is valid as the second argument to the towctrans() function. Otherwise, it returns 0.

Example that uses wctrans()

This example translates the lowercase alphabet to uppercase, and back to lowercase.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <wchar.h>
#include <wctype.h>
int main()
   char *alpha = "abcdefghijklmnopqrstuvwxyz";
  char *tocase[2] = {"toupper", "tolower"};
  wchar t *wcalpha;
  int i, j;
  size_t alphalen;
  alphalen = strlen(alpha)+1;
  wcalpha = (wchar_t *)malloc(sizeof(wchar_t)*alphalen);
  mbstowcs(wcalpha, alpha, 2*alphalen);
   for (i=0; i<2; ++i) {
     printf("Input string: %ls\n", wcalpha);
      for (j=0; j
     for (j=0; j
```

Related Information

- "towctrans() Translate Wide Character" on page 378
- "<wctype.h>" on page 18

wctype() — Get Handle for Character Property Classification

Format

```
#include <wctype.h>
wctype_t wctype(const char *property);
```

Language Level: XPG4

Description

The wctype() function is defined for valid property names as defined in the current locale. The property is a string that identifies a generic character class for which locale specific type information is required. The function returns a value of type wctype_t, which can be used as the second argument to a call of the iswctype() function.

The wctype() function determines values of wctype_t according to rules of the coded character set that are defined by character type information in the program's locale (category LC_CTYPE). Values that are returned by the wctype() are valid until a call to setlocale() that changes the category LC_CTYPE.

Return Value

The wctype() function returns zero if the given property name is not valid for the current locale (category LC_CTYPE or LC_UCS2_CTYPE). Otherwise it returns a value of type wctype_t that can be used in calls to iswctype().

Example that uses wctype()

```
#include <wchar.h>
#define
          UPPER LIMIT
                         0xFF
      int main(void)
         int
              WC;
                                        UPPER LIMIT; wc++)
         for
               (wc
                    =
                          0;
                              WC <=
            printf("%#4x ",
                              wc);
            printf("%c",
                                                                        : "
                           iswctype(wc,
                                           wctype("print")) ?
                                                                  WC
");
            printf("%s",
                           iswctype(wc,
                                           wctype("alnum")) ?
                                                                  "AN"
");
                                                                  "A"
            printf("%s",
                                           wctype("alpha")) ?
                           iswctype(wc,
");
                                                                        : "
                                                                  "B"
            printf("%s",
                           iswctype(wc,
                                           wctype("blank")) ?
");
            printf("%s",
                                           wctype("cntrl")) ?
                                                                  "C"
                           iswctype(wc,
");
            printf("%s",
                                           wctype("digit")) ?
                                                                  "D"
                            iswctype(wc,
");
            printf("%s",
                                           wctype("graph")) ?
                                                                  "G"
                            iswctype(wc,
");
            printf("%s",
                            iswctype(wc,
                                           wctype("lower")) ?
                                                                  "L"
");
                                                                  "PU"
            printf("%s",
                            iswctype(wc,
                                           wctype("punct")) ?
");
                                                                  "S"
            printf("%s",
                           iswctype(wc,
                                           wctype("space")) ?
");
            printf("%s",
                           iswctype(wc,
                                           wctype("print")) ?
                                                                  "PR"
");
                                                                  "U"
            printf("%s",
                                           wctype("upper")) ?
                           iswctype(wc,
");
            printf("%s",
                            iswctype(wc,
                                           wctype("xdigit")) ?
                                                                  пХп
                                                                        : "
");
            putchar('\n');
         return
                  0;
            The
                  output
                           should
                                          similar
            0x1f
            0x20
                                                S
                                                      PR
            0x21
                                       G
                                               PU
                                                        PR
                                               PU
                                                        PR
            0x22
                                       G
            0x23
                                               PU
                                                        PR
                                       G
                                               PU
            0x24
                   $
                                       G
                                                        PR
            0x25
                                       G
                                               PU
                                                        PR
            0x26
                                       G
                                               PU
                                                        PR
                                                        PR
            0x27
                                       G
                                               PU
            0x28
                                       G
                                               PU
                                                        PR
                                               PU
                                                        PR
            0x29
                                       G
            0x2a
                                       G
                                               PU
                                                        PR
            0x2b
                                       G
                                               PU
                                                        PR
            0x2c
                                        G
                                                PU
                                                        PR
                                       G
                                                        PR
            0x2d
                                               PU
            0x2e
                                        G
                                                         PR
            0x2f
                                        G
                                                PU
                                                         PR
                                           G
                   0
                       AN
                                      D
                                                          PR
                                                                  Χ
            0x30
            0x31
                   1
                       AN
                                      D
                                           G
                                                          PR
                                                                  Χ
                   2
                                           G
                                                                  Χ
            0x32
                       ΑN
                                      D
            0x33
                   3
                       AN
                                      D
                                           G
                                                          PR
                                                                  Χ
            0x34
                   4
                        ΑN
                                      D
                                           G
                                                          PR
                                                                  χ
                                                          PR
                                                                  χ
            0x35
                                           G
                       AN
                                      D
      }
```

Related Information

- "<wchar.h>" on page 18
- "<wctype.h>" on page 18

wcwidth() — Determine the Display Width of a Wide Character

Format

```
#include <wchar.h>
int wcwidth (const wint_t wc);
```

Language Level: XPG4

Thread Safe: YES.

Description

The wcwidth() function determines the number of printing positions that a graphic representation of *wc* occupies on a display device. Each of the printing wide characters occupies its own number of printing positions on a display device. The number is independent of its location on the device.

Note: This function is accessible only if LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) is specified on the compilation command.

Return Value

The wcwidth() function either returns:

- 0, if wc is a null wide character; or
- the number of printing position occupied by wc; or
- -1, if wc is not a printing wide character.

The behavior of the wcwidth() function is affected by the LC_CTYPE category of the current locale. If the program is compiled with LOCALETYPE(*LOCALEUCS2), the wide character properties are those defined by the LC_UCS2_CTYPE category of the current locale.

Example that uses wcwidth()

Related Information

- "wcswidth() Determine the Display Width of a Wide Character String" on
- "<wchar.h>" on page 18

wfopen() —Open Files

Format

```
#include <ifs.h>
FILE * wfopen(const wchar_t *filename,const wchar_t *mode);
```

Language Level: ILE C Extension

Thread Safe: Yes

Description

The wfopen() function works like the fopen() function, except:

- wfopen() accepts file name and mode as wide characters.
- wfopen() assumes CCSID 13488 if neither CCSID nor codepage keyword is specified.

The wfopen() function is made available by specifying SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALEUCS2) on the compilation command.

wmemchr() —Locate Wide Character in Wide-Character Buffer

Format

```
#include <wchar.h>
wchar t *wmemchr(const wchar t *s, wchar t c, size t n);
```

Language Level: ANSI

Description

The wmemchr() function locates the first occurrence of c in the initial n wide characters of the object pointed to by s. If n has the value 0, the wmemchr() function finds no occurrence of *c*, and returns a NULL pointer.

Return Value

The wmemchr() function returns a pointer to the located wide character, or a NULL pointer if the wide character does not occur in the object.

Example that uses wmemchr()

This example finds the first occurrence of 'A' in the wide-character string.

```
#include <stdio.h>
#include <wchar.h>

main()
{
    wchar_t *in = L"1234ABCD";
    wchar_t *ptr;
    wchar_t fnd = L'A';

    printf("\nEXPECTED: ABCD");
    ptr = wmemchr(in, L'A', 6);
    if (ptr == NULL)
        printf("\n** ERROR ** ptr is NULL, char L'A' not found\n");
    else
        printf("\nRECEIVED: %ls \n", ptr);
}
```

Related Information

- "memchr() Search Buffer" on page 187
- "strchr() Search for Character" on page 325
- "wcschr() Search for Wide Character" on page 402
- "wmemcmp() —Compare Wide-Character Buffers"
- "wmemcpy() —Copy Wide-Character Buffer" on page 444
- "wmemmove() Copy Wide-Character Buffer" on page 445
- "wmemset() Set Wide Character Buffer to a Value" on page 446
- "<wchar.h>" on page 18

wmemcmp() —Compare Wide-Character Buffers

Format

```
#include <wchar.h>
int wmemcmp(const wchar_t *s1, const wchar_t *s2, size_t n);
```

Language Level: ANSI

Description

The wmemcmp() function compares the first n wide characters of the object pointed to by s1 to the first n wide characters of the object pointed to by s2. If n has the value 0, the wmemcmp() function returns 0.

Return Value

The wmemcmp() function returns a value according to the relationship between the two strings, s1 and s2:

Integer Value	Meaning
Less than 0	s1 less than s2
0	s1 equal to s2
Greater than 0	s1 greater than s2

Example that uses wmemcmp()

This example compares the wide-character string in to out using the wmemcmp()function.

```
#include <wchar.h>
#include <stdio.h>
#include <locale.h>
main()
  int rc;
  wchar t * in = L"12345678";
  wchar t *out = L"12AAAAAB";
  setlocale(LC ALL, "POSIX");
  printf("\nGREATER is the expected result");
  rc = wmemcmp(in, out, 3);
  if (rc == 0)
     printf("\nArrays are EQUAL %ls %ls \n", in, out);
  else
  if (rc > 0)
     printf("\nArray %ls GREATER than %ls \n", in, out);
  else
     printf("\nArray %ls LESS than %ls \n", in, out);
  /*****************
     The output should be:
     GREATER is the expected result
     Array 12345678 GREATER than 12AAAAAB
  ************************************
```

Related Information

- "memcmp() Compare Buffers" on page 188
- "strcmp() Compare Strings" on page 326
- "wcscmp() Compare Wide-Character Strings" on page 403
- "wcsncmp() Compare Wide-Character Strings" on page 411
- "wmemchr() —Locate Wide Character in Wide-Character Buffer" on page 442
- "wmemcpy() —Copy Wide-Character Buffer"
- "wmemmove() Copy Wide-Character Buffer" on page 445
- "wmemset() Set Wide Character Buffer to a Value" on page 446
- "<wchar.h>" on page 18

wmemcpy() —Copy Wide-Character Buffer

Format

```
#include <wchar.h>
wchar_t *wmemcpy(wchar_t *s1, const wchar_t *s2, size_t n);
```

Language Level: ANSI

Description

The wmemcpy() function copies n wide characters from the object pointed to by s2to the object pointed to by s1. If s1 and s2 overlap, the result of the copy is unpredictable. If n has the value 0, the wmemcpy() function copies 0 wide characters.

Return Value

The wmemcpy() function returns the value of s1.

Example that uses wmemcpy()

This example copies the first four characters from out to in. In the expected output, the first four characters in both strings will be "ABCD".

```
#include <wchar.h>
#include <stdio.h>

main()
{
    wchar_t *in = L"12345678";
    wchar_t *out = L"ABCDEFGH";
    wchar_t *ptr;

    printf("\nExpected result: First 4 chars of in change");
    printf(" and are the same as first 4 chars of out");
    ptr = wmemcpy(in, out, 4);
    if (ptr == in)
        printf("\nArray in %ls array out %ls \n", in, out);
    else
    {
    printf("\n*** ERROR ***");
    printf(" returned pointer wrong");
    }
}
```

Related Information

- "memcpy() Copy Bytes" on page 189
- "strcpy() Copy Strings" on page 330
- "strncpy() Copy Strings" on page 346
- "wcscpy() Copy Wide-Character Strings" on page 405
- "wcsncpy() Copy Wide-Character Strings" on page 413
- "wmemchr() —Locate Wide Character in Wide-Character Buffer" on page 442
- "wmemcmp() —Compare Wide-Character Buffers" on page 443
- "wmemmove() Copy Wide-Character Buffer"
- "wmemset() Set Wide Character Buffer to a Value" on page 446
- "<wchar.h>" on page 18

wmemmove() — Copy Wide-Character Buffer

Format

```
#include <wchar.h>
wchar t *wmemmove(wchar t *s1, const wchar t *s2, size t n);
```

Language Level: ANSI

Description

The wmemmove() function copies n wide characters from the object pointed to by s2 to the object pointed to by s1. Copying takes place as if the n wide characters from the object pointed to by s2 are first copied into a temporary array, of n wide characters, that does not overlap the objects pointed to by s1 or s2. Then, the

wmemmove() function copies the n wide characters from the temporary array into the object pointed to by s1. If n has the value 0, the wmemmove() function copies 0 wide characters.

Return Value

The wmemmove() function returns the value of *s*1.

Example that uses wmemmove()

This example copies the first five characters in a string to overlay the last five characters in the same string. Since the string is only nine characters long, the source and target overlap.

```
#include <wchar.h>
#include <stdio.h>
void main()
  wchar_t *theString = L"ABCDEFGHI";
  printf("\nThe original string: %ls \n", theString);
  wmemmove(theString+4, theString, 5);
  printf("\nThe string after wmemmove: %ls \n", theString);
  return;
  /**********************************
     The output should be:
     The original string: ABCDEFGHI
     The string after wmemmove: ABCDABCDE
  ***********************************
```

Related Information

- "memmove() Copy Bytes" on page 191
- "wmemchr() —Locate Wide Character in Wide-Character Buffer" on page 442
- "wmemcpy() —Copy Wide-Character Buffer" on page 444
- "wmemcmp() —Compare Wide-Character Buffers" on page 443
- "wmemset() Set Wide Character Buffer to a Value"
- "<wchar.h>" on page 18

wmemset() — Set Wide Character Buffer to a Value

Format

```
#include <wchar.h>
wchar t *wmemset(wchar t *s, wchar t c, size t n);
```

Language Level: ANSI

Description

The wmemset () function copies the value of c into each of the first n wide characters of the object pointed to by s. If n has the value 0, the wmemset() function copies 0 wide characters.

Return Value

The wmemset() function returns the value of s.

Example that uses wmemset()

This example sets the first 6 wide characters to the wide character 'A'.

```
#include <wchar.h>
#include <stdio.h>

void main()
{
    wchar_t *in = L"1234ABCD";
    wchar_t *ptr;

    printf("\nEXPECTED: AAAAAACD");
    ptr = wmemset(in, L'A', 6);
    if (ptr == in)
        printf("\nResults returned - %ls \n", ptr);
    else
        {
            printf("\n** ERROR ** wrong pointer returned\n");
        }
}
```

Related Information

- "memset() Set Bytes to Value" on page 192
- "wmemchr() —Locate Wide Character in Wide-Character Buffer" on page 442
- "wmemcpy() —Copy Wide-Character Buffer" on page 444
- "wmemcmp() —Compare Wide-Character Buffers" on page 443
- "wmemmove() Copy Wide-Character Buffer" on page 445
- "<wchar.h>" on page 18

wprintf() — Format Data as Wide Characters and Print

Format

```
#include <wchar.h>
int wprintf(const wchar_t *format,...);
```

Language Level: ANSI

Thread Safe: YES.

Description

```
A wprintf(format, ...) is equivalent to fwprintf(stdout, format, ...).
```

Note: This function is available only when SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALE) are specified on the compilation command.

Return Value

The wprintf() function returns the number of wide characters transmitted. If an output error occurred, the wprintf() function returns a negative value.

Example that uses wprintf()

This example prints the wide character a. Date and time may be formatted according to your locale's representation. The output goes to stdout.

```
#include <wchar.h>
#include <stdarg.h>
#include <locale.h>
int main(void)
  setlocale(LC_ALL, "POSIX");
  wprintf (L"%c\n", L'a');
  return(0);
/* A long 'a' is written to stdout */
```

Related Information

- "printf() Print Formatted Characters" on page 200
- "btowc() Convert Single Byte to Wide Character" on page 54
- "mbrtowc() Convert a Multibyte Character to a Wide Character (Restartable)" on page 176
- "vfwprintf() Format Argument Data as Wide Characters and Write to a Stream" on page 387
- "fwprintf() Format Data as Wide Characters and Write to a Stream" on page 123
- "vswprintf() Format and Write Wide Characters to Buffer" on page 393
- "<wchar.h>" on page 18

wscanf() — Read Data Using Wide-Character Format String

Format

```
#include <wchar.h>
int wscanf(const wchar_t *format,...);
```

Language Level: ANSI

Description

The wscanf() function is equivalent to the fwscanf() function with the argument stdin interposed before the arguments of the wscanf() function.

Note: This function is available only when SYSIFCOPT(*IFSIO) and LOCALETYPE(*LOCALE) or LOCALETYPE(*LOCALEUCS2) are specified on the compilation command.

Return Value

If an input failure occurs before any conversion, the wscanf() function returns the value of the macro EOF.

Otherwise, the wscanf() function returns the number of input items assigned. It can be fewer than provided for, or even zero, in the event of an early matching failure.

Example that uses wscanf()

This example scans various types of data.

```
#include <stdio.h>
#include <wchar.h>
int main(void)
  int i;
  float fp;
  char c,s[81];
  printf("Enter an integer, a real number, a character and a string : \n");
  if (wscanf(L"%d %f %c %s", &i, &fp,&c, s) != 4)
     printf("Not all of the fields were not assigned\n");
  else {
     printf("integer = %d\n", i);
     printf("real number = %f\n", fp);
printf("character = %c\n", c);
     printf("string = %s\n", s);
  return 0;
  /************************************
     The output should be similar to:
     Enter an integer, a real number, a character and a string :
     12 2.5 a yes
     integer = 12
     real number = 2.500000
     character = a
     string = yes
```

Related Information

- "fwprintf() Format Data as Wide Characters and Write to a Stream" on page 123
- "swprintf() Format and Write Wide Characters to Buffer" on page 367
- "wprintf() Format Data as Wide Characters and Print" on page 447
- "scanf() Read Data" on page 299
- "sscanf() Read Data" on page 322
- "swscanf() Read Wide Character Data" on page 369
- "fscanf() Read Formatted Data" on page 113
- "fwscanf() Read Data from Stream Using Wide Character" on page 128
- "<wchar.h>" on page 18

Chapter 3. Run-Time Considerations

This chapter provides information on:

- Exception and condition management
- Interlanguage data type compatibility
- CCSID (Coded Character Set Identifier) source file conversion

errno Macros

The following table lists which error macros the ILE C library functions can set.

Table 12. errno Macros

Error Macro	Description	Set by Function
EBADDATA	The message data is not valid.	perror, strerror
EBADF	The catalog descriptor is not valid.	catclose, catgets, clearerr, fgetc, fgetpos, fgets, fileno, freopen, fseek, fsetpos, getc, rewind
EBADKEYLN	The key length specified is not valid.	_Rreadk, _Rlocate
EBADMODE	The file mode specified is not valid.	fopen, freopen, _Ropen
EBADNAME	Bad file name specified.	fopen, freopen, _Ropen
EBADPOS	The position specified is not valid.	fsetpos
EBADSEEK	Bad offset for a seek operation.	fgetpos, fseek
EBUSY	The record or file is in use.	perror, strerror
ECONVERT	Conversion error.	wcstomb, wcswidth
EDOM	Domain error in math function.	acos, asin, atan2, cos, exp, fmod, gamma, hypot, j0, j1, jn, y0, y1, yn, log, log10, pow, sin, strtol, strtoul, sqrt, tan
EGETANDPUT	An illegal read operation occurred after a write operation.	fgetc, fread, getc, getchar
EILSEQ	The character sequence does not form a valid multibyte character.	fgetwc, fgetws, getwc, mblen,mbrlen, mbrtowc, mbsrtowcs, mbstowcs, mbtowc, printf, scanf, ungetwc, wcrtomb, wcsrtombs, wcstombs, wctomb, wcswidth, wcwidth
EINVAL	The signal is not valid.	printf, scanf, signal, swprintf, swscanf, wcstol, wcstoll, wcstoul, wcstoull

Table 12. errno Macros (continued)

Error Macro	Description	Set by Function
EIO	Consecutive calls of I/O occurred.	I/O
EIOERROR	A non-recoverable I/O error occurred.	All I/O functions
EIORECERR	A recoverable I/O error occurred.	All I/O functions
ENODEV	Operation attempted on a wrong device.	fgetpos, fsetpos, fseek, ftell, rewind
ENOENT	File or library is not found.	perror, strerror
ENOPOS	No record at specified position.	fsetpos
ENOREC	Record not found.	fread, perror, strerror
ENOTDLT	File is not opened for delete operations.	_Rdelete
ENOTOPEN	File is not opened.	clearerr, fclose, fflush, fgetpos, fopen, freopen, fseek, ftell, setbuf, setvbuf, _Ropen, _Rclose
ENOTREAD	File is not opened for read operations.	fgetc, fread, ungetc, _Rreadd, _Rreadf, _Rreadindv, _Rreadk, _Rreadl, _Rreadn, _Rreadnc, _Rreadp, _Rreads, _Rlocate
ENOTUPD	File is not opened for update operations.	_Rrlslck, _Rupdate
ENOTWRITE	File is not opened for write operations.	fputc, fwrite, _Rwrite, _Rwrited, _Rwriterd
ENUMMBRS	More than 1 member.	ftell
ENUMRECS	Too many records.	ftell
EPAD	Padding occurred on a write operation.	fwrite
EPERM	Insufficient authorization for access.	perror, strerror
EPUTANDGET	An illegal write operation occurred after a read operation.	fputc, fwrite, fputs, putc, putchar
ERANGE	Range error in math function.	cos, cosh, gamma, exp, j0, j1, jn, y0, y1, yn, log, log10, ldexp, pow, sin, sinh, strtod, strtol, strtoul, tan, wcstol, wcstoll, wcstoul, wcstoul, wcstod
ERECIO	File is opened for record I/O, so character-at-a-time processing functions cannot be used.	fgetc, fgetpos, fputc, fread, fseek, fsetpos, ftell
ESTDERR	stderr cannot be opened.	feof, ferror, fgetpos, fputc, fseek, fsetpos, ftell, fwrite

Table 12. errno Macros (continued)

Error Macro	Description	Set by Function
ESTDIN	stdin cannot be opened.	fgetc, fgetpos, fread, fseek, fsetpos, ftell
ESTDOUT	stdout cannot be opened.	fgetpos, fputc, fseek, fsetpos, ftell, fwrite
ETRUNC	Truncation occurred on I/O operation.	Any I/O function that reads or writes a record sets errno to ETRUNC.

errno Values for Integrated File System Enabled C Stream I/O

The following table describes the possible settings when using integrated file system enabled stream I/O.

Table 13. errno Values for IFS Enabled C Stream I/O

C Stream Function	Possible errno Values
clearerr	EBADF
fclose	EAGAIN, EBADF, EIO, EUNKNOWN
feof	EBADF
ferror	EBADF
fflush	EACCES, EAGAIN, EBADF, EBUSY, EDAMAGE, EFAULT, EFBIG, EINVAL, EIO, ENOMEM, ENOSPC, ETRUNC, EUNKNOWN, EPUTANDGET, ENOTWRITE, EPAD
fgetc	EBADF, EACCES, EAGAIN, EBUSY, EDAMAGE, EFAULT, EINVAL, EIO, ENOMEM, EUKNOWN, EGETANDPUT, EDOM, ENOTREAD,
fgetpos	EACCESS, EAGAIN, EBADF, EBUSY, EDAMAGE, EFAULT, EINVAL, EIO, ENOSYSRSC, EUNATCH, EUNKNOWN
fgets	EBADF, EACCES, EAGAIN, EBUSY, EDAMAGE, EFAULT, EINVAL, EIO, ENOMEM, EUKNOWN, EGETANDPUT, EDOM, ENOTREAD
fgetwc	EBADF, EILSEQ
fgetws	EBADF, EILSEQ
fopen	EAGAIN, EBADNAME, EBADF, ECONVERT, EDAMAGE, EEXITS, EFAULT, EINVAL, EIO, EISDIR, ELOOP, ENOENT, ENOMEM, ENOSPC, ENOSYS, ENOSYSRSC, ENOTDIR
fprintf	EACCES, EAGAIN, EBADF, EBUSY, EDAMAGE, EFAULT, EFBIG, EINVAL, EIO, ENOMEM, ENOSPC, ETRUNC, EUNKNOWN, EPUTANDGET, ENOTWRITE, EPAD
fputc	EACCES, EAGAIN, EBADF, EBUSY, EDAMAGE, EFAULT, EFBIG, EINVAL, EIO, ENOMEM, ENOSPC, ETRUNC, EUNKNOWN, EPUTANDGET, ENOTWRITE, EPAD
fputs	EACCES, EAGAIN, EBADF, EBUSY, EDAMAGE, EFAULT, EFBIG, EINVAL, EIO, ENOMEM, ENOSPC, ETRUNC, EUNKNOWN, EPUTANDGET, ENOTWRITE, EPAD
fread	EBADF, EACCES, EAGAIN, EBUSY, EDAMAGE, EFAULT, EINVAL, EIO, ENOMEM, EUKNOWN, EGETANDPUT, EDOM, ENOTREAD

Table 13. errno Values for IFS Enabled C Stream I/O (continued)

C Stream Function	Possible errno Values
freopen	EACCES, EAGAIN, EBADNAME, EBADF, EBUSY, ECONVERT, EDAMAGE, EEXITS, EFAULT, EINVAL, EIO, EISDIR, ELOOP, EMFILE, ENAMETOOLONG, ENFILE, ENOENT, ENOMEM, ENOSPC, ENOSYS, ENOSYSRSC, ENOTDIR
fscanf	EBADF, EACCES, EAGAIN, EBUSY, EDAMAGE, EFAULT, EINVAL, EIO, ENOMEM, EUKNOWN, EGETANDPUT, EDOM, ENOTREAD
fseek	EACCES, EAGAIN, EBADF, EBUSY, EDAMAGE, EINVAL, EIO, ENOENT, ENOSPC, ENOSYSRSC, ESPIPE, EUNKNOWN, EFAULT, EPERM, EUNATCH, EUNKNOWN
fsetpos	EACCES, EAGAIN, ABADF, EBUSY, EDAMAGE, EINVAL, EIO, ENOENT, ENOSPC, ENOSYSRSC, ESPIPE, EUNKNOWN, EFAULT, EPERM, EUNATCH, EUNKNOWN
ftell	EACCESS, EAGAIN, EBADF, EBUSY, EDAMAGE, EFAULT, EINVAL, EIO, ENOSYSRSC, EUNATCH, EUNKNOWN
fwrite	EACCESS, EAGAIN, EBADF, EBUSY, EDAMAGE, EFAULT, EINVAL, EIO, ENOSYSRSC, EUNATCH, EUNKNOWN
getc	EBADF, EACCES, EAGAIN, EBUSY, EDAMAGE, EFAULT, EINVAL, EIO, ENOMEM, EUKNOWN, EGETANDPUT, EDOM, ENOTREAD
getchar	EBADF, EACCES, EAGAIN, EBUSY, EDAMAGE, EFAULT, EINVAL, EIO, ENOMEM, EUKNOWN, EGETANDPUT, EDOM, ENOTREAD
gets	EBADF, EACCES, EAGAIN, EBUSY, EDAMAGE, EFAULT, EINVAL, EIO, ENOMEM, EUKNOWN, EGETANDPUT, EDOM, ENOTREAD
getwc	EBADF, EILSEQ
perror	EBADF
printf	EACCES, EAGAIN, EBADF, EBUSY, EDAMAGE, EFAULT, EFBIG, EILSEQ, EINVAL, EIO, ENOMEM, ENOSPC, ETRUNC, EUNKNOWN, EPUTANDGET, ENOTWRITE, EPAD
putc	EACCES, EAGAIN, EBADF, EBUSY, EDAMAGE, EFAULT, EFBIG, EINVAL, EIO, ENOMEM, ENOSPC, ETRUNC, EUNKNOWN, EPUTANDGET, ENOTWRITE, EPAD
putchar	EACCES, EAGAIN, EBADF, EBUSY, EDAMAGE, EFAULT, EFBIG, EINVAL, EIO, ENOMEM, ENOSPC, ETRUNC, EUNKNOWN, EPUTANDGET, ENOTWRITE, EPAD
puts	EACCES, EAGAIN, EBADF, EBUSY, EDAMAGE, EFAULT, EFBIG, EINVAL, EIO, ENOMEM, ENOSPC, ETRUNC, EUNKNOWN, EPUTANDGET, ENOTWRITE, EPAD
remove	EACCES, EAGAIN, EBADNAME, EBADF, EBUSY, ECONVERT, EDAMAGE, EEXITS, EFAULT, EINVAL, EIO, EISDIR, ELOOP, ENAMETOOLONG, ENOENT, ENOMEM, ENOSPC, ENOTDIR, EPERM, EROOBJ, EUNKNOWN, EXDEV
rename	EACCES, EAGAIN, EBADNAME, EBUSY, ECONVERT, EDAMAGE, EEXIST, EFAULT, EINVAL, EIO, EISDIR, ELOOP, ENAMETOOLONG, ENOTEMPTY, ENOENT, ENOMEM, ENOSPC, ENOTDIR, EMLINK, EPERM, EUNKNOWN, EXDEV
rewind	EACCES, EAGAIN, EBADF, EBUSY, EDAMAGE, EINVAL, EIO, ENOENT, ENOSPC, ENOSYSRSC, ESPIPE, EUNKNOWN, EFAULT, EPERM, EUNATCH, EUNKNOWN

Table 13. errno Values for IFS Enabled C Stream I/O (continued)

C Stream Function	Possible errno Values	
scanf	EBADF, EACCES, EAGAIN, EBUSY, EDAMAGE, EFAULT, EILSEQ, EINVAL, EIO, ENOMEM, EUKNOWN, EGETANDPUT, EDOM, ENOTREAD	
setbuf	EBADF, EINVAL, EIO	
setvbuf	EBADF, EINVAL, EIO	
tmpfile	EACCES, EAGAIN, EBADNAME, EBADF, EBUSY, ECONVERT, EDAMAGE, EEXITS, EFAULT, EINVAL, EIO, EISDIR, ELOOP, EMFILE, ENAMETOOLONG, ENFILE, ENOENT, ENOMEM, ENOSPC, ENOSYS, ENOSYSRSC, ENOTDIR, EPERM, EROOBJ, EUNKNOW N, EXDEV	
tmpnam	EACCESS, EAGAIN, EBADF, EBUSY, EDAMAGE, EFAULT, EINVAL, EIO, ENOENT, ENOSYSRSC, EUNATCH, EUNKNOWN	
ungetc	EBADF, EIO	
ungetwc	EBADF, EILSEQ	
vfprintf	EACCES, EAGAIN, EBADF, EBUSY, EDAMAGE, EFAULT, EFBIG, EINVAL, EIO, ENOMEM, ENOSPC, ETRUNC, EUNKNOWN, EPUTANDGET, ENOTWRITE, EPAD	
vprintf	EACCES, EAGAIN, EBADF, EBUSY, EDAMAGE, EFAULT, EFBIG, EINVAL, EIO, ENOMEM, ENOSPC, ETRUNC, EUNKNOWN, EPUTANDGET, ENOTWRITE, EPAD	

Record Input and Output Error Macro to Exception Mapping

The following table describes what occurs if the signal SIGIO is raised. Only *ESCAPE, *NOTIFY, and *STATUS messages are monitored.

Table 14. Record Input and Output Error Macro to Exception Mapping

Description	Messages (_EXCP_MSGID)	errno setting
*STATUS and *NOTIFY	CPF4001 to CPF40FF, CPF4401 to CPF44FF, CPF4901 to CPF49FF, CPF5004	errno is not set, a default reply is returned to the operating system.
Recoverable I/O error	CPF4701 to CPF47FF, CPF4801 to CPF48FF, CPF5001 to CPF5003, CPF5005 to CPF50FF,	EIORECERR
Non-recoverable I/O error ²	CPF4101 to CPF41FF, CPF4201 to CPF42FF, CPF4301 to CPF43FF, CPF4501 to CPF45FF, CPF4601 to CPF46FF, CPF5101 to CPF51FF, CPF5201 to CPF52FF, CPF5301 to CPF53FF, CPF5401 to CPF54FF, CPF5501 to CPF55FF, CPF5601 to CPF56FF	EIOERROR
Truncation occurred at I/O operation	C2M3003	ETRUNC
File is not opened	C2M3004	ENOTOPEN

Description	Messages (_EXCP_MSGID)	errno setting
File is not opened for read operations	C2M3005	ENOTREAD
File is not opened for write operations	C2M3009	ENOTWRITE
Bad file name specified	C2M3014	EBADNAME
The file mode specified is not valid	C2M3015	EBADMODE
File is not opened for update operations	C2M3041	ENOTUPD
File is not opened for delete operations	C2M3042	ENOTDLT
The key length specified is not valid	C2M3044	EBADKEYLN
A non-recoverable I/O error occurred	C2M3101	EIOERROR
A recoverable I/O error occurred	C2M3102	EIORECERR

Note:

- ¹ The error is percolated to the user, therefore the user's direct monitor handlers, ILE C condition handlers and signal handler may get control. The initial setting for SIGIO is SIG_IGN.
- ² The type of device determines whether the error is recoverable or not recoverable. The following IBM publications contain information about recoverable and non-recoverable system exceptions for each specific file type:
 - ICF Programming
 - ADTS/400: Advanced Printer Function
 - Application Display Programming
 - **Database Programming**
 - Tape and Diskette Device Programming

Signal Handling Action Definitions

The following table shows the initial state of the C signal values and their handling action definitions when SYSIFCOPT(*NOASYNCSIGNAL) is specified on the compilation command. SIG_DFL always percolates the condition to the handler. Resume indicates the exception is handled, and the application continues.

Table 15. Handling Action Definitions for Signal Values

Signal Value	Initial State	SIG_DFL	SIG_IGN	Return from Handler
SIGABRT ¹	SIG_DFL	Percolate	Ignore	Resume
SIGALL ²	SIG_DFL	Percolate	Ignore	Resume
SIGFPE	SIG_DFL	Percolate	Ignore ³	Resume ⁴
SIGILL	SIG_DFL	Percolate	Ignore ³	Resume ⁴
SIGINT	SIG_DFL	Percolate	Ignore	Resume
SIGIO	SIG_IGN	Percolate	Ignore	Resume
SIGOTHER	SIG_DFL	Percolate	Ignore ³	Resume ⁴

Table 15. Handling Action Definitions for Signal Values (continued)

Signal Value	Initial State	SIG_DFL	SIG_IGN	Return from Handler
SIGSEGV	SIG_DFL	Percolate	Ignore ³	Resume ⁴
SIGTERM	SIG_DFL	Percolate	Ignore	Resume
SIGUSR1	SIG_DFL	Percolate	Ignore	Resume
SIGUSR2	SIG_DFL	Percolate	Ignore	Resume

Note:

- ¹ Can only be signaled by the raise() function or the abort() function
- ullet 2 SIGALL cannot be signaled by the raise() function.
- ³ If the value of the signal is SIGFPE, SIGILL or SIGSEGV the behavior is undefined.
- $\, \bullet \,\,$ If the signal is hardware-generated, then the behavior undefined.

The following table shows the initial state of the C signal values and their handling action definitions with SYSIFCOPT(*ASYNCSIGNAL) is specified on the compilation command.

Table 16. Default Actions for Signal Values

Value	Default Action	Meaning
SIGABRT	2	Abnormal termination.
SIGFPE	2	Arithmetic exceptions that are not masked, such as overflow, division by zero, and incorrect operation.
SIGILL	2	Detection of an incorrect function image.
SIGINT	2	Interactive attention.
SIGSEGV	2	Incorrect access to storage.
SIGTERM	2	Termination request sent to the program.
SIGUSR1	2	Intended for use by user applications.
SIGUSR2	2	Intended for use by user applications.
SIGALRM	2	A timeout signal that is sent by alarm().
SIGHUP	2	A controlling terminal is hung up, or the controlling process ended.
SIGKILL	1	A termination signal that cannot be caught or ignored.
SIGPIPE	3	A write to a pipe that is not being read.
SIGQUIT	2	A quit signal for a terminal.
SIGCHLD	3	An ended or stopped child process. SIGCLD is an alias name for this signal.
SIGCONT	5	If stopped, continue.
SIGSTOP	4	A stop signal that cannot be caught or ignored.
SIGTSTP	4	A stop signal for a terminal.
SIGTTIN	4	A background process attempted to read from a controlling terminal.
SIGTTOU	4	A background process attempted to write to a controlling terminal.
SIGIO	3	Completion of input or output.
		-

Table 16. Default Actions for Signal Values (continued)

SIGURG	3	High bandwidth data is available at a socket.
SIGPOLL	2	Pollable event.
SIGBUS	2	Specification exception.
SIGPRE	2	Programming exception.
SIGSYS	2	Bad system call.
SIGTRAP	2	Trace or breakpoint trap.
SIGPROF	2	Profiling timer expired.
SIGVTALRM	2	Virtual timer expired.
SIGXCPU	2	Processor time limit exceeded.
SIGXFSZ	2	File size limit exceeded.
SIGDANGER	2	System crash is imminent.
SIGPCANCEL	2	Thread termination signal that cannot be caught or ignored.

Default Actions:

- 1 End the process immediately
- End the request.
- 3 Ignore the signal.
- 4 Stop the process.
- 5 Continue the process if it is currently stopped. Otherwise, ignore the

Signal to iSeries Exception Mapping

The following table shows the system exception messages that are mapped to a signal. All *ESCAPE exception messages are mapped to signals. The *STATUS and *NOTIFY messages that map to SIGIO as defined in Table 14 on page 455 are mapped to signals.

Table 17. Signal to iSeries Exception Mapping

Signal	Message
SIGABRT	C2M1601
SIGALL	C2M1610 (if explicitly raised)
SIGFPE	C2M1602, MCH1201 to MCH1204, MCH1206 to MCH1215, MCH1221 to MCH1224, MCH1838 to MCH1839
SIGILL	C2M1603, MCH0401, MCH1002, MCH1004, MCH1205, MCH1216 to MCH1219, MCH1801 to MCH1802, MCH1807 to MCH1808, MCH1819 to MCH1820, MCH1824 to MCH1825, MCH1832, MCH1837, MCH1852, MCH1854 to MCH1857, MCH1867, MCH2003 to MCH2004, MCH2202, MCH2602, MCH2604, MCH2808, MCH2810 to MCH2811, MCH3201 to MCH3203, MCH4201 to MCH4211, MCH4213, MCH4296 to MCH4298, MCH4401 to MCH4403, MCH4406 to MCH4408, MCH4421, MCH4427 to MCH4428, MCH4801, MCH4804 to MCH4805, MCH5001 to MCH5003, MCH5401 to MCH5402, MCH5601, MCH6001 to MCH6002, MCH6201, MCH6208, MCH6216, MCH6220, MCH6403, MCH6601 to MCH6602, MCH6609 to MCH6612
SIGINT	C2M1604

Table 17. Signal to iSeries Exception Mapping (continued)

Signal	Message	
SIGIO	C2M1609, See Table 14 on page 455 for the exception mappings.	
SIGOTHER	C2M1611 (if explicitly raised)	
SIGSEGV	C2M1605, MCH0201, MCH0601 to MCH0606, MCH0801 to MCH0803, MCH1001, MCH1003, MCH1005 to MCH1006, MCH1220, MCH1401 to MCH1402, MCH1602, MCH1604 to MCH1605, MCH1668, MCH1803 to MCH1806, MCH1809 to MCH1811, MCH1813 to MCH1815, MCH1821 to MCH1823, MCH1826 to MCH1829, MCH1833, MCH1836, MCH1848, MCH1850, MCH1851, MCH1864 to MCH1866, MCH1898, MCH2001 to MCH2002, MCH2005 to MCH2006, MCH2201, MCH2203 to MCH2205, MCH2401, MCH2601, MCH2603, MCH2605, MCH2801 to MCH2804, MCH2806 to MCH2809, MCH3001, MCH3401 to MCH3408, MCH3410, MCH3601 to MCH3602, MCH3603 to MCH3604, MCH3802, MCH4001 to MCH4002, MCH4010, MCH4212, MCH4404 to MCH4405, MCH4416 to MCH4402, MCH4422 to MCH4426, MCH4429 to MCH4437, MCH4601, MCH4802 to MCH4803, MCH4806 to MCH4812, MCH5201 to MCH5204, MCH5602 to MCH5603, MCH5801 to MCH5804, MCH6203 to MCH6204, MCH6206, MCH6217 to MCH6219, MCH6221 to MCH6222, MCH6401 to MCH6402, MCH6404, MCH6603 to MCH6608, MCH6801	
SIGTERM	C2M1606	
SIGUSR1	C2M1607	
SIGUSR2	C2M1608	

Cancel Handler Reason Codes

The following table lists the bits that are set in the reason code. If the activation group is to be stopped, then the activation group is stopped bit is also set in the reason code. These bits must be correlated to _CNL_MASK_T in _CNL_Hndlr_Parms_T in <except.h>. Column 2 contains the macro constant defined for the cancel reason mask in <except.h>.

Table 18. Determining Canceled Invocation Reason Codes

Function	Bits set in reason code	Rationale
Library routines		
exit	_EXIT_VERB	The definition of exit is normal end of processing, and therefore invocations canceled by this function is done with a reason code of <i>normal</i> .
abort	_ABNORMAL_TERM _EXIT_VERB	The definition of abort is abnormal end of processing, and therefore invocations canceled by this function are done with a reason code of <i>abnormal</i> .
longjmp	_JUMP	The general use of the longjmp() function is to return from an exception handler, although it may be used in non-exception situations as well. It is used as part of the "normal" path for a program, and therefore any invocations canceled because of it are cancelled with a reason code of normal.
Unhandled function check	_ABNORMAL_TERM_ UNHANDLED_EXCP	Not handling an exception which is an abnormal situation.

Table 18. Determining Canceled Invocation Reason Codes (continued)

Function	Bits set in reason code	Rationale
System APIs		
CEEMRCR	_ABNORMAL_TERM _EXCP_SENT	This API is only used during exception processing. It is typically used to cancel invocations where a resume is not possible, or at least the behavior would be undefined if control was resumed in them. Also, these invocations have had a chance to handle the exception but did not do so. Invocations canceled by this API are done with reason code of abnormal.
QMHSNDPM /QMHRSNEM (escape messages) Message Handler APIs	_ABNORMAL_TERM _EXCP_SENT	All invocations down to the target invocation are canceled without any chance of handling the exception. The API topic contains information on these APIs.
iSeriescommands		
Process end	_ABNORMAL_TERM _PROCESS_TERM _AG_TERMINATING	Any externally initiated shutdown of an activation group is considered abnormal.
RCLACTGRP	_ABNORMAL_TERM _RCLRSC	The default is abnormal termination. The termination could be normal if a normal/abnormal flag is added to the command.

Table 19. Common Reason Code for Cancelling Invocations

Bit	Description	Header File Constant <except.h></except.h>
Bits 0	Reserved	CACCPUITS
Bits 1	Invocation canceled due to sending exception message	_EXCP_SENT
Bits 2-15	Reserved	
Bit 16	0 - normal end of process 1 - abnormal end of process	_ABNORMAL_TERM
Bit 17	Activation Group is ending.	_AG_TERMINATING
Bit 18	Initiated by Reclaim Activation Group (RCLACTGRP)	_RCLRSC
Bit 19	Initiated by the process end.	_PROCESS_TERM
Bit 20	Initiated by an exit() function.	_EXIT_VERB
Bit 21	Initiated by an unhandled function check.	_UNHANDLED_EXCP
Bit 22	Invocation canceled due to a longjmp() function.	_JUMP
Bit 23	Invocation canceled due to a jump because of exception processing.	_JUMP_EXCP
Bits 24-31	Reserved (0)	

Exception Classes

In a CL program, you can monitor for a selected group of exceptions, or a single exception, based on the exception identifier. The only class2 values the exception handler will monitor for are _C2_MH_ESCAPE, _C2_MH_STATUS, _C2_MH_NOTIFY, and _C2_MH_FUNCTION_CHECK. For more information about using the #pragma exception handler directive, see the WebSphere Development Studio: ILE C/C++ Programmer's Guide. This table defines all the exception classes you can specify.

Table 20. Exception Classes

Bit position	Header File Constant in <except.h></except.h>	Exception class
0	_C1_BINARY_OVERFLOW	Binary overflow or divide by zero
1	_C1_DECIMAL_OVERFLOW	Decimal overflow or divide by zero
2	_C1_DECIMAL_DATA_ERROR	Decimal data error
3	_C1_FLOAT_OVERFLOW	Floating-point overflow or divide by zero
4	_C1_FLOAT_UNDERFLOW	Floating-point underflow or inexact result
5	_C1_INVALID_FLOAT_OPERAND	Floating-point invalid operand or conversion error
6	_C1_OTHER_DATA_ERROR	Other data error, for example edit mask
7	_C1_SPECIFICATION_ERROR	Specification (operand alignment) error
8	_C1_POINTER_NOT_VALID	Pointer not set/pointer type invalid
9	_C1_OBJECT_NOT_FOUND	Object not found
10	_C1_OBJECT_DESTROYED	Object destroyed
11	_C1_ADDRESS_COMP_ERROR	Address computation underflow or overflow
12	_C1_SPACE_ALLOC_ERROR	Space not allocated at specified offset
13	_C1_DOMAIN_OR_STATE_VIOLATION	Domain/State protection violation
14	_C1_AUTHORIZATION_VIOLATION	Authorization violation
15	_C1_JAVA_THROWN_CLASS	Exception thrown for a Java class.
16-28	_C1_VLIC_RESERVED	VLIC reserved
29	_C1_OTHER_MI_EXCEPTION	Remaining MI-generated exceptions (other than function check)
30	_C1_MI_GEN_FC_OR_MC	MI-generated function check or machine check
31	_C1_MI_SIGEXP_EXCEPTION	Message generated via Signal Exception instruction

Table 20. Exception Classes (continued)

Bit position	Header File Constant in <except.h></except.h>	Exception class
32-39	n/a	reserved
40	_C2_MH_ESCAPE	*ESCAPE
41	_C2_MH_NOTIFY	*NOTIFY
42	_C2_MH_STATUS	*STATUS
43	_C2_MH_FUNCTION_CHECK	function check
44-63	n/a	reserved

Data Type Compatibility

Each high-level language has different data types. When you want to pass data between programs that are written in different languages, you must be aware of these differences.

Some data types in the ILE C programming language have no direct equivalent in other languages. However, you can simulate data types in other languages that use ILE C data types.

The following table shows the ILE C data type compatibility with ILE RPG.

Table 21. ILE C Data Type Compatibility with ILE RPG

ILE C declaration in prototype	ILE RPG D spec, columns 33 to 39	Length	Comments
char[n] char *	nA	n	An array of characters where n=1 to 32766.
char	1A	1	An Indicator that is a variable starting with *IN.
char[n]	nS 0	n	A zoned decimal.
char[2n]	nG	2n	A graphic added.
char[2n+2]	Not supported.	2n+2	A graphic data type.
_Packed struct {short i; char[n]}	Not supported.	n+2	A variable length field where i is the intended length and n is the maximum length.
char[n]	D	8, 10	A date field.
char[n]	Т	8	A time field.
char[n]	Z	26	A timestamp field.
short int	51 0	2	An integer field.
short unsigned int	5U 0	2	An unsigned integer field.
int	101 0	4	An integer field.
unsigned int	10U 0	4	An unsigned integer field
long int	101 0	4	An integer field.
long unsigned int	10U 0	4	An unsigned integer field.

Table 21. ILE C Data Type Compatibility with ILE RPG (continued)

ILE C declaration in prototype	ILE RPG D spec, columns 33 to 39	Length	Comments
struct {unsigned int : n}x;	Not supported.	4	A 4-byte unsigned integer, a bitfield.
float	Not supported.	4	A 4-byte floating point.
double	Not supported.	8	An 8-byte double.
long double	Not supported.	8	An 8-byte long double.
enum	Not supported.	1, 2, 4	Enumeration.
*	*	16	A pointer.
decimal(n,p)	nP p	n/2+1	A packed decimal. n must be less than or equal to 30.
union.element	<type> with keyword OVERLAY(longest field)</type>	element length	An element of a union.
data_type[n]	<type> with keyword DIM(n)</type>	16	An array to which C passes a pointer.
struct	data structure	n	A structure. Use the _Packed qualifier on the struct.
pointer to function	*	16	A 16-byte pointer.
	with keyword PROCPTR		

The following table shows the ILE C data type compatibility with ILE COBOL.

Table 22. ILE C Data Type Compatibility with ILE COBOL

ILE C declaration in prototype	ILE COBOL LINKAGE SECTION	Length	Comments
char[n] char *	PIC X(n).	n	An array of characters where n=1 to 3,000,000
char	PIC 1 INDIC	1	An indicator.
char[n]	PIC S9(n) DISPLAY	n	A zoned decimal.
wchar_t[n]	PIC G(n)	2n	A graphic data type.
_Packed struct {short i; char[n]}	05 VL-FIELD. 10 i PIC S9(4) COMP-4. 10 data PIC X(n).	n+2	A variable length field where i is the intended length and n is the maximum length.
char[n]	PIC X(n).	6	A date field.
char[n]	PIC X(n).	5	A day field.
char	PIC X.	1	A day-of-week field.
char[n]	PIC X(n).	8	A time field.
char[n]	PIC X(n).	26	A time stamp field.
short int	PIC S9(4) COMP-4.	2	A 2-byte signed integer with a range of -9999 to +9999.
short int	PIC S9(4) BINARY.	2	A 2-byte signed integer with a range of -9999 to +9999.

Table 22. ILE C Data Type Compatibility with ILE COBOL (continued)

ILE C declaration in prototype	ILE COBOL LINKAGE SECTION	Length	Comments
int	PIC S9(9) COMP-4.	4	A 4-byte signed integer with a range of -999999999 to +999999999.
int	PIC S9(9) BINARY.	4	A 4-byte signed integer with a range of -999999999 to +999999999.
int	USAGE IS INDEX	4	A 4-byte integer.
long int	PIC S9(9) COMP-4.	4	A 4-byte signed integer with a range of -999999999 to +999999999.
long int	PIC S9(9) BINARY.	4	A 4-byte signed integer with a range of -999999999 to +999999999.
struct {unsigned int : n}x;	PIC 9(9) COMP-4. PIC X(4).	4	Bitfields can be manipulated using hex literals.
float	Not supported.	4	A 4-byte floating point.
double	Not supported.	8	An 8-byte double.
long double	Not supported.	8	An 8-byte long double.
enum	Not supported.	1, 2, 4	Enumeration.
*	USAGE IS POINTER	16	A pointer.
decimal(n,p)	PIC S9(n-p)V9(p) COMP-3	n/2+1	A packed decimal.
decimal(n,p)	PIC S9(n-p) 9(p) PACKED-DECIMAL	n/2+1	A packed decimal.
union.element	REDEFINES	element length	An element of a union.
data_type[n]	OCCURS	16	An array to which C passes a pointer.
struct	01 record 05 field1 05 field2	n	A structure. Use the _Packed qualifier on the struct. Structures passed should be passed as a pointer to the structure if you want to change the contents of the structure.
pointer to function	PROCEDURE-POINTER	16	A 16 byte pointer to a procedure.
Not supported.	PIC S9(18) COMP-4.	8	An 8 byte integer.
Not supported.	PIC S9(18) BINARY.	8	An 8 byte integer.

The following table shows the ILE C data type compatibility with ILE CL. Table 23. ILE C Data Type Compatibility with ILE CL

ILE C declaration in			
prototype	CL	Length	Comments
char[n] char *	*CHAR LEN(&N)	n	An array of characters where n=1 to 32766. A null-terminated string. For example, CHGVAR &V1 VALUE (&V *TCAT X'00') where &V1 is one byte bigger than &V.
char	*LGL	1	Holds '1' or '0'.
_Packed struct {short i; char[n]}	Not supported.	n+2	A variable length field where i is the intended length and n is the maximum length.
integer types	Not supported.	1, 2, 4	A 1-, 2-, or 4- byte signed or unsigned integer.
float constants	CL constants only.	4	A 4- or 8- byte floating point.
decimal(n,p)	*DEC	n/2+1	A packed decimal. The limit of n is 15 and p is 9.
union.element	Not supported.	element length	An element of a union.
struct	Not supported.	n	A structure. Use the _Packed qualifier on the struct.
pointer to function	Not supported.	16	A 16-byte pointer.

The following table shows the ILE C data type compatibility with OPM RPG/ 400° . Table 24. ILE C Data Type Compatibility with OPM RPG/400

ILE C declaration in prototype	OPM RPG/400 I spec, DS subfield columns spec	Length	Comments
char[n] char *	1 10	n	An array of characters where n=1 to 32766.
char	*INxxxx	1	An Indicator that is a variable starting with *IN.
char[n]	1 nd (d>=0)	n	A zoned decimal. The limit of n is 30.
char[2n+2]	Not supported.	2n+2	A graphic data type.
_Packed struct {short i; char[n]}	Not supported.	n+2	A variable length field where i is the intended length and n is the maximum length.
char[n]	Not supported.	6, 8, 10	A date field.
char[n]	Not supported.	8	A time field.
char[n]	Not supported.	26	A time stamp field.
short int	B 1 20	2	A 2-byte signed integer with a range of -9999 to +9999.
int	B 1 40	4	A 4-byte signed integer with a range of -999999999 to +999999999.

Table 24. ILE C Data Type Compatibility with OPM RPG/400 (continued)

ILE C declaration in prototype	OPM RPG/400 I spec, DS subfield columns spec	Length	Comments
long int	B 1 40	4	A 4-byte signed integer with a range of -999999999 to +999999999.
struct {unsigned int : n}x;	Not supported.	4	A 4-byte unsigned integer, a bitfield.
float	Not supported.	4	A 4-byte floating point.
double	Not supported.	8	An 8-byte double.
long double	Not supported.	8	An 8-byte long double.
enum	Not supported.	1, 2, 4	Enumeration.
*	Not supported.	16	A pointer.
decimal(n,p)	P 1 n/2+1d	n/2+1	A packed decimal. n must be less than or equal to 30.
union.element	data structure subfield	element length	An element of a union.
data_type[n]	E-SPEC array	16	An array to which C passes a pointer.
struct	data structure	n	A structure. Use the _Packed qualifier on the struct.
pointer to function	Not supported.	16	A 16 byte pointer.

The following table shows the ILE C data type compatibility with OPM COBOL/400 $\!\!^{\odot}$.

Table 25. ILE C Data Type Compatibility with OPM COBOL/400

ILE C declaration in prototype	OPM COBOL LINKAGE SECTION	Length	Comments
char[n] char *	PIC X(n).	n	An array of characters where n=1 to 3,000,000
char	PIC 1 INDIC	1	An indicator.
char[n]	PIC S9(n) USAGE IS DISPLAY	n	A zoned decimal. The limit of n is 18.
_Packed struct {short i; char[n]}	05 VL-FIELD. 10 i PIC S9(4) COMP-4. 10 data PIC X(n).	n+2	A variable length field where i is the intended length and n is the maximum length.
char[n]	PIC X(n).	6, 8, 10	A date field.
char[n]	PIC X(n).	8	A time field.
char[n]	PIC X(n).	26	A time stamp field.
short int	PIC S9(4) COMP-4.	2	A 2 byte signed integer with a range of -9999 to +9999.
int	PIC S9(9) COMP-4.	4	A 4-byte signed integer with a range of -999999999 to +999999999.
long int	PIC S9(9) COMP-4.	4	A 4-byte signed integer with a range of -999999999 to +999999999.

Table 25. ILE C Data Type Compatibility with OPM COBOL/400 (continued)

ILE C declaration in prototype	OPM COBOL LINKAGE SECTION	Length	Comments
struct {unsigned int : n}x;	PIC 9(9) COMP-4. PIC X(4).	4	Bitfields can be manipulated using hex literals.
float	Not supported.	4	A 4-byte floating point.
double	Not supported.	8	An 8-byte double.
long double	Not supported.	8	An 8-byte long double.
enum	Not supported.	1, 2, 4	Enumeration.
*	USAGE IS POINTER	16	A pointer.
decimal(n,p)	PIC S9(n-p)V9(p) COMP-3	n/2+1	A packed decimal. The limits of n and p are 18.
union.element	REDEFINES	element length	An element of a union.
data_type[n]	OCCURS	16	An array to which C passes a pointer.
struct	01 record	n	A structure. Use the _Packed qualifier on the struct. Structures passed should be passed as a pointer to the structure if you want to change the contents of the structure.
pointer to function	Not supported.	16	A 16-byte pointer.
Not supported.	PIC S9(18) COMP-4.	8	An 8 byte integer.

The following table shows the ILE \ensuremath{C} data type compatibility with \ensuremath{CL} .

Table 26. ILE C Data Type Compatibility with CL

ILE C declaration in prototype	CL	Length	Comments
char[n] char *	*CHAR LEN(&N)	n	An array of characters where n=1 to 32766. A null terminated string. For example, CHGVAR &V1 VALUE (&V *TCAT X'00') where &V1 is one byte bigger than &V. The limit of n is 9999.
char	*LGL	1	Holds '1' or '0'.
_Packed struct {short i; char[n]}	Not supported.	n+2	A variable length field where i is the intended length and n is the maximum length.
integer types	Not supported.	1, 2, 4	A 1-, 2- or 4- byte signed or unsigned integer.
float constants	CL constants only.	4	A 4- or 8- byte floating point.
decimal(n,p)	*DEC	n/2+1	A packed decimal. The limit of n is 15 and p is 9.
union.element	Not supported.	element length	An element of a union.
struct	Not supported.	n	A structure. Use the _Packed qualifier on the struct.

Table 26. ILE C Data Type Compatibility with CL (continued)

ILE C declaration in prototype	CL	Length	Comments
pointer to function	Not supported.	16	A 16-byte pointer.

The following table shows how arguments are passed from a command line CL call to an ILE C program.

Table 27. Arguments Passed From a Command Line CL Call to an ILE C Program

Command Line Argument	Argv Array	ILE C Arguments
	argv[0]	"LIB/PGMNAME"
	argv[1255]	normal parameters
′123.4′	argv[1]	"123.4"
123.4	argv[2]	0000000123.40000D
'Hi'	argv[3]	"Hi"
Lo	argv[4]	"LO"

A CL character array (string) will not be NULL-ended when passed to an ILE C program. A C program that will receive such arguments from a CL program should not expect the strings to be NULL-ended. You can use the QCMDEXC to ensure that all the arguments will be NULL-ended.

The following table shows how CL constants are passed from a compiled CL program to an ILE C program.

Table 28. CL Constants Passed from a Compiled CL Program to an ILE C Program

Compile CL Program		
Argument	Argv Array	ILE C Arguments
	argv[0]	"LIB/PGMNAME"
	argv[1255]	normal parameters
′123.4′	argv[1]	"123.4"
123.4	argv[2]	0000000123.40000D
'Hi'	argv[3]	"Hi"
Lo	argv[4]	"LO"

A command processing program (CPP) passes CL constants as defined in Table 28. You define an ILE C program as a command processing program when you create your own CL command with the Create Command (CRTCMD) command to call the ILE C program.

The following table shows how CL variables are passed from a compiled CL program to an ILE C program. All arguments are passed by reference from CL to

Table 29. CL Variables Passed from a Compiled CL Program to an ILE C Program

CL Variables	ILE C Arguments
DCL VAR(&v) TYPE(*CHAR) LEN(10) VALUE('123.4')	123.4

Table 29. CL Variables Passed from a Compiled CL Program to an ILE C Program (continued)

CL Variables	ILE C Arguments
DCL VAR(&d) TYPE(*DEC) LEN(10) VALUE(123.4)	0000000123.40000D
DCL VAR(&h) TYPE(*CHAR) LEN(10) VALUE('Hi')	Hi
DCL VAR(&i) TYPE(*CHAR) LEN(10) VALUE(Lo)	LO
DCL VAR(&j) TYPE(*LGL) LEN(1) VALUE('1')	1

CL variables and numeric constants are not passed to an ILE C program with null-ended strings. Character constants and logical literals are passed as null-ended strings, but are not padded with blanks. Numeric constraints such as packed decimals are passed as 15,5 (8 bytes).

Run-time Character Set

Each EBCDIC CCSID consists of two character types: invariant characters and variant characters.

The following table identifies the hexadecimal representation of the invariant characters in the C character set.

Table 30. Invariant Characters

0x4b	< 0x4c	(0x4d	+ 0x4e	& 0x50	* 0x5c) 0x5d	; 0x5e
- 0x60	¦ 0x6a	, 0x6b	% 0x6c	_ 0x6d	> 0x6e	? 0x6f	: 0x7a
@ 0x7c	, 0x7d	= 0x7e	" 0x7f	a-i 0x81 - 0x89	j-r 0x91 - 0x99	s-z 0xa2 - 0xa9	A-I 0xc1 - 0xc9
J-R 0xd1 - 0xd9	S-Z 0xe2 - 0xe9	0-9 0xf0 - 0xf9	'\a' 0x2f	'\b' 0x16	'\t' 0x05	'\v' 0x0b	'\f' 0x0c
'\r' 0x0d	'\n' 0x15	0x40					

The following table identifies the hexadecimal representation of the variant characters in the C character set for the most commonly used CCSIDs.

Table 31. Variant Characters in Different CCSIDs

CC- SID	I	!	7	\	,	#	~	[]	^	{	}	/	¢	\$
037	0x4f	0x5a	0x5f	0xe0	0x79	0x7b	0xa1	0xba	0xbb	0xb0	0xc0	0xd0	0x61	0x4a	0x5b
256	0xbb	0x4f	0xba	0xe0	0x79	0x7b	0xa1	0x4a	0x5a	0x5f	0xc0	0xd0	0x61	0xb0	0x5b
273	0xbb	0x4f	0xba	0xec	0x79	0x7b	0x59	0x63	0xfc	0x5f	0x43	0xdc	0x61	0xb0	0x5b
277	0xbb	0x4f	0xba	0xe0	0x79	0x4a	0xdc	0x9e	0x9f	0x5f	0x9c	0x47	0x61	0xb0	0x67
278	0xbb	0x4f	0xba	0x71	0x51	0x63	0xdc	0xb5	0x9f	0x5f	0x43	0x47	0x61	0xb2	0x67
280	0xbb	0x4f	0xba	0x48	0xdd	0xb1	0x58	0x90	0x51	0x5f	0x44	0x45	0x61	0xb0	0x5b

Table 31. Variant Characters in Different CCSIDs (continued)

CC- SID	I	!	г	\	`	#	~	[]	^	{	}	/	¢	\$
284	0x4f	0xbb	0x5f	0xe0	0x79	0x69	0xbd	0x4a	0x5a	0xba	0xc0	0xd0	0x61	0xb0	0x5b
285	0x4f	0x5a	0x5f	0xe0	0x79	0x7b	0xbc	0xb1	0xbb	0xba	0xc0	0xd0	0x61	0xb0	0x4a
297	0xbb	0x4f	0xba	0x48	0xa0	0xb1	0xbd	0x90	0x65	0x5f	0x51	0x54	0x61	0xb0	0x5b
500	0xbb	0x4f	0xba	0xe0	0x79	0x7b	0xa1	0x4a	0x5a	0x5f	0xc0	0xd0	0x61	0xb0	0x5b

See the Globalization topic for more information on coding variant characters in the other IBM CCSIDs.

Asynchronous Signal Model

The Asynchronous Signal Model (ASM) is used when the SYSIFCOPT(*ASYNCSIGNAL) option is specified on the the compilation. It is inteded for campatibility with applications ported from the UNIX operating system. In the ASM model, the signal() and raise() functions are implemented using the OS/400 Signal APIs described in the API topic under the Programming heading in the Information Center.

OS/400 exceptions sent to an ASM program are converted to asynchronous signals. The exceptions are processed by an asynchronous signal handler

Modules compiled to use the ASM can be intermixed with modules using the Original Signal Model (OSM) in the same processes, programs, and service programs. This is true even when SYSIFCOPT(*ASYNCSIGNAL) is not specified on the compilation command for programs compiled with the OSM. There are several differences between the two signal models:

- The OSM is based on exceptions, while the ASM is based on asynchronous signals.
- Under the OSM, the signal vector and signal mask are scoped to the activation group. Under ASM, there is one signal vector per process and one signal mask per thread. Both types of signal vectors and signal masks are maintained during compilation.
- The same asynchronous signal vector and signal masks are operated on by all ASM modules in a thread, regardless of the activation group the signal vector and signal masks belong to. You should save and restore the state of the signal vector and signal masks to ensure that they are not changed by any ASM modules. The OSM does not use the asynchronous signal vector and signal
- Signals raised by OSM programs will be sent as exceptions. Under the OSM, the exceptions are received and handled by the _C_exception_router function, which directly invokes the OSM signal handler of the the user.

Asynchronous signals are not mapped to exceptions, and are not handled by signal handlers registered under the OSM. Under the ASM, the exceptions are received and handled by the _C_async_exception_router function, which maps the exception to an asynchronous signal, and uses the SNDSIG MI instruction to raise an asynchronous signal. An ASM signal handler receives control from the OS/400 asynchonous signal component.

When an OSM program raises a signal, the generated exception percolates up the invocation stack until it finds an exception monitor. If the prior invocation is an OSM function, the _C_exception_router catches the exception and performs the OSM signal action. The ASM signal handler does not receive the signal.

If the prior invocation is an ASM function, the _C_async_exception_router handles the exception, maps to an asynchronous signal, and raises it using the MI SNDSIG instruction. The handling of the asynchronous signal then depends on the asynchronous signal vector and mask state of the thread, as defined in the OS/400 Signal Management topic.

If the OSM program raising the signal is running in the same activation group with the ASM program, the exception will be mapped to an asynchronous signal using the mapping described previously. The signal ID is preserved when the exception is mapped to a signal. So, signal handlers registered with the asynchronous signal model will be able to receive signals generated under the original signal model. Use this approach to integrate two programs using different signal models.

If a program or service program using the OSM is running a different activation group, any signal that is unmonitored in that activation group will cause the termination of that program and activation group. The unmonitored signal is then percolated to the calling program as a CEE9901 exception. The CEE9901 exception is mapped to a SIGSEGV asynchronous signal.

Note: This approach is not recommended.

- Under the ASM, the C functionsraise() and signal() are integrated with the system signal functions, such as kill() and sigaction(). These two sets of APIs can be used interchangeably. This cannot be done under the OSM.
- A user-specified exception monitor established with #pragma_exception_handler
 has precedent over the compiler-generated monitor, which invokes
 _C_async_exception_router. This precedence enables you to bypass the
 asynchronous signal generating exception monitor in some situations.
- The _GetExcData() function is not available under the ASM to retrieve the exception ID associated with the signal. If an extended signal handler is established using the sigaction() function, however, it can access the exception information from the signal-specific data structure. For more information, see "_GetExcData() Get Exception Data" on page 135.

Part 2. Appendixes

Appendix A. Library Functions and Extensions

This chapter summarizes all the standard C library functions and the ILE Clibrary extensions.

Standard C Library Functions Table, By Name

This table briefly describes the C library functions, listed in alphabetical order. This table provides the include file name and the function prototype for each function.

Table 32. Standard C Library Functions

Function	System Include File	Function Prototype	Description
abort	stdlib.h	void abort(void);	Stops a program abnormally.
abs	stdlib.h	int abs(int <i>n</i>);	Calculates the absolute value of an integer argument <i>n</i> .
acos	math.h	double acos(double <i>x</i>);	Calculates the arc cosine of x .
asctime	time.h	char *asctime(const struct tm *time);	Converts the <i>time</i> that is stored as a structure to a character string.
asctime_r	time.h	char *asctime_r (const struct tm *tm, char *buf);	Converts <i>tm</i> that is stored as a structure to a character string. (Restartable version of asctime.)
asin	math.h	double asin(double x);	Calculates the arc sine of x .
assert	assert.h	void assert(int expression);	Prints a diagnostic message and ends the program if the expression is false.
atan	math.h	double atan(double x);	Calculates the arc tangent of x .
atan2	math.h	double atan2(double <i>y</i> , double <i>x</i>);	Calculates the arc tangent of y/x .
atexit	stdlib.h	int atexit(void (*func)(void));	Registers a function to be called at normal termination.
atof	stdlib.h	double atof(const char *string);	Converts <i>string</i> to a double-precision floating-point value.
atoi	stdlib.h	int atoi(const char *string);	Converts string to an integer.
atol	stdlib.h	long int atol(const char *string);	Converts <i>string</i> to a long integer.
bsearch	stdlib.h	void *bsearch(const void *key, const void *base, size_t num, size_t size, int (*compare) (const void *element1, const void *element2));	Performs a binary search on an array of <i>num</i> elements, each of <i>size</i> bytes. The array must be sorted in ascending order by the function pointed to by <i>compare</i> .

Table 32. Standard C Library Functions (continued)

Function	System Include File	Function Prototype	Description
btowc	stdio.h wchar.h	wint_t btowc(int c);	Determines whether <i>c</i> constitues a valid multibyte character in the initial shift state.
calloc	stdlib.h	void *calloc(size_t num, size_t size);	Reserves storage space for an array of <i>num</i> elements, each of size <i>size</i> , and initializes the values of all elements to 0.
catclose ⁶	nl_types.h	int catclose (nl_catd catd);	Closes a previously opened message catalog.
catgets ⁶	nl_types.h	char *catgets(nl_catd catd, int set_id, int msg_id, const char *s);	Retrieves a message from an open message catalog.
catopen ⁶	nl_types.h	nl_catd catopen (const char *name, int oflag);	Opens a message catalog, which must be done before a message can be retrieved.
ceil	math.h	double ceil(double <i>x</i>);	Calculates the double value representing the smallest integer that is greater than or equal to <i>x</i> .
clearerr	stdio.h	<pre>void clearerr(FILE *stream);</pre>	Resets the error indicators and the end-of-file indicator for <i>stream</i> .
clock	time.h	clock_t clock(void);	Returns the processor time that has elapsed since the job was started.
cos	math.h	double cos(double <i>x</i>);	Calculates the cosine of <i>x</i> .
cosh	math.h	double cosh(double <i>x</i>);	Calculates the hyperbolic cosine of <i>x</i> .
ctime	time.h	char *ctime(const time_t *time);	Converts <i>time</i> to a character string.
ctime_r	time.h	char *ctime_r(const time_t *timer, char *buf);	Converts <i>timer</i> to a character string. (Restartable version of ctime.)
difftime	time.h	double difftime(time_t time2, time_t time1);	Computes the difference between <i>time2</i> and <i>time1</i> .
div	stdlib.h	div_t div(int numerator, int denominator);	Calculates the quotient and remainder of the division of numerator by denominator.
erf	math.h	double erf(double <i>x</i>);	Calculates the error function of <i>x</i> .
erfc	math.h	double erfc(double <i>x</i>);	Calculates the error function for large values of x .
exit	stdlib.h	void exit(int status);	Ends a program normally.
exp	math.h	double exp(double <i>x</i>);	Calculates the exponential function of a floating-point argument <i>x</i> .

Table 32. Standard C Library Functions (continued)

Function	System Include File	Function Prototype	Description
fabs	math.h	double fabs(double <i>x</i>);	Calculates the absolute value of a floating-point argument <i>x</i> .
fclose	stdio.h	int fclose(FILE *stream);	Closes the specified stream.
fdopen ⁵	stdio.h	FILE *fdopen(int handle, const char *type);	Associates an input or output stream with the file identified by handle.
feof	stdio.h	int feof(FILE *stream);	Tests whether the end-of-file flag is set for a given <i>stream</i> .
ferror	stdio.h	int ferror(FILE *stream);	Tests for an error indicator in reading from or writing to stream.
fflush ¹	stdio.h	int fflush(FILE *stream);	Writes the contents of the buffer associated with the output <i>stream</i> .
fgetc ¹	stdio.h	int fgetc(FILE *stream);	Reads a single unsigned character from the input stream.
fgetpos ¹	stdio.h	int fgetpos(FILE *stream, fpos_t *pos);	Stores the current position of the file pointer associated with <i>stream</i> into the object pointed to by <i>pos</i> .
fgets ¹	stdio.h	char *fgets(char *string, int n, FILE *stream);	Reads a string from the input stream.
fgetwc ⁶	stdio.h wchar.h	wint_t fgetwc(FILE *stream);	Reads the next multibyte character from the input stream pointed to by <i>stream</i> .
fgetws ⁶	stdio.h wchar.h	wchar_t *fgetws(wchar_t *wcs, int n, FILE *stream);	Reads wide characters from the stream into the array pointed to by <i>wcs</i> .
fileno ⁵	stdio.h	int fileno(FILE *stream);	Determines the file handle currently associated with stream.
floor	math.h	double floor(double <i>x</i>);	Calculates the floating-point value representing the largest integer less than or equal to <i>x</i> .
fmod	math.h	double fmod(double <i>x</i> , double <i>y</i>);	Calculates the floating-point remainder of x/y .
fopen	stdio.h	FILE *fopen(const char *filename, const char *mode);	Opens the specified file.
fprintf	stdio.h	<pre>int fprintf(FILE *stream, const char *format-string, arg-list);</pre>	Formats and prints characters and values to the output stream.
fputc ¹	stdio.h	<pre>int fputc(int c, FILE *stream);</pre>	Prints a character to the output <i>stream</i> .
fputs ¹	stdio.h	int fputs(const char *string, FILE *stream);	Copies a string to the output stream.

Table 32. Standard C Library Functions (continued)

Function	System Include File	Function Prototype	Description
fputwc ⁶	stdio.h wchar.h	wint_t fputwc(wchar_t wc, FILE *stream);	Converts the wide character <i>wc</i> to a multibyte character and writes it to the output stream pointed to by <i>stream</i> at the current position.
fputws ⁶	stdio.h wchar.h	int fputws(const wchar_t *wcs, FILE *stream);	Converts the wide-character string <i>wcs</i> to a multibyte-character string and writes it to <i>stream</i> as a multibyte character string.
fread	stdio.h	size_t fread(void *buffer, size_t size, size_t count, FILE *stream);	Reads up to <i>count</i> items of <i>size</i> length from the input <i>stream</i> , and stores them in <i>buffer</i> .
free	stdlib.h	void free(void *ptr);	Frees a block of storage.
freopen	stdio.h	FILE *freopen(const char *filename, const char *mode, FILE *stream);	Closes <i>stream</i> , and reassigns it to the file specified.
frexp	math.h	double frexp(double <i>x</i> , int * <i>expptr</i>);	Separates a floating-point number into its mantissa and exponent.
fscanf	stdio.h	<pre>int fscanf(FILE *stream, const char *format-string, arg-list);</pre>	Reads data from <i>stream</i> into locations given by <i>arg-list</i> .
fseek ¹	stdio.h	<pre>int fseek(FILE *stream, long int offset, int origin);</pre>	Changes the current file position associated with <i>stream</i> to a new location.
fsetpos ¹	stdio.h	int fsetpos(FILE *stream, const fpos_t *pos);	Moves the current file position to a new location determined by <i>pos</i> .
ftell ¹	stdio.h	long int ftell(FILE *stream);	Gets the current position of the file pointer.
fwide ⁶	stdio.h wchar.h	<pre>int fwide(FILE *stream, int mode);</pre>	Determines the orientation of the stream pointed to by <i>stream</i> .
fwprintf ⁶	stdio.h wchar.h	<pre>int fwprintf(FILE *stream, const wchar_t *format, arg-list);</pre>	Writes output to the stream pointed to by <i>stream</i> .
fwrite	stdio.h	size_t fwrite(const void *buffer, size_t size,size_t count, FILE *stream);	Writes up to <i>count</i> items of <i>size</i> length from <i>buffer</i> to <i>stream</i> .
fwscanf ⁶	stdio.h wchar.h	<pre>int fwscanf(FILE *stream, const wchar_t *format, arg-list)</pre>	Reads input from the stream pointed to by <i>stream</i> .
gamma	math.h	double gamma(double <i>x</i>);	Computes the Gamma Function
getc ¹	stdio.h	int getc(FILE *stream);	Reads a single character from the input <i>stream</i> .
getchar ¹	stdio.h	int getchar(void);	Reads a single character from stdin.

Table 32. Standard C Library Functions (continued)

Function	System Include File	Function Prototype	Description
getenv	stdlib.h	char *getenv(const char *varname);	Searches environment variables for <i>varname</i> .
gets	stdio.h	char *gets(char *buffer);	Reads a string from <i>stdin</i> , and stores it in <i>buffer</i> .
getwc ⁶	stdio.h wchar.h	wint_t getwc(FILE *stream);	Reads the next multibyte character from <i>stream</i> , converts it to a wide character and advances the associated file position indicator for <i>stream</i> .
getwchar ⁶	wchar.h	wint_t getwchar(void);	Reads the next multibyte character from stdin, converts it to a wide character, and advances the associated file position indicator for stdin.
gmtime	time.h	struct tm *gmtime(const time_t *time);	Converts a <i>time</i> value to a structure of type tm.
gmtime_r	time.h	struct tm *gmtime_r (const time_t *timer, struct tm *result);	Converts a <i>timer</i> value to a structure of type tm. (Restartable version of gmtime.)
hypot	math.h	double hypot(double side1, double side2);	Calculates the hypotenuse of a right-angled triangle with sides of length <i>side1</i> and <i>side2</i> .
isalnum	ctype.h	int isalnum(int c);	Tests if c is alphanumeric.
isalpha	ctype.h	int isalpha(int c);	Tests if c is alphabetic.
isascii	ctype.h	int isascii(int c);	Tests if <i>c</i> is within the 7-bit US-ASCII range.
iscntrl	ctype.h	int iscntrl(int c);	Tests if c is a control character.
isdigit	ctype.h	int isdigit(int c);	Tests if c is a decimal digit.
isgraph	ctype.h	int isgraph(int c);	Tests if <i>c</i> is a printable character excluding the space.
islower	ctype.h	int islower(int c);	Tests if c is a lowercase letter.
isprint	ctype.h	int isprint(int c);	Tests if <i>c</i> is a printable character including the space.
ispunct	ctype.h	int ispunct(int c);	Tests if <i>c</i> is a punctuation character.
isspace	ctype.h	int isspace(int c);	Tests if <i>c</i> is a whitespace character.
isupper	ctype.h	int isupper(int c);	Tests if <i>c</i> is an uppercase letter.
iswalnum ⁴	wctype.h	int iswalnum (wint_t wc);	Checks for any alphanumeric wide character.
iswalpha ⁴	wctype.h	int iswalpha (wint_t wc);	Checks for any alphabetic wide character.
iswcntrl ⁴	wctype.h	int iswcntrl (wint_t wc);	Tests for any control wide character.

Table 32. Standard C Library Functions (continued)

Function	System Include File	Function Prototype	Description
iswctype ⁴	wctype.h	<pre>int iswctype(wint_t wc, wctype_t wc_prop);</pre>	Determines whether or not the wide character wc has the property wc_prop.
iswdigit ⁴	wctype.h	int iswdigit (wint_t wc);	Checks for any decimal-digit wide character.
iswgraph ⁴	wctype.h	int iswgraph (wint_t wc);	Checks for any printing wide character except for the wide-character space.
iswlower ⁴	wctype.h	int iswlower (wint_t wc);	Checks for any lowercase wide character.
iswprint ⁴	wctype.h	int iswprint (wint_t wc);	Checks for any printing wide character.
iswpunct ⁴	wctype.h	int iswpunct (wint_t wc);	Test for a wide non-alphanumeric, non-space character.
iswspace ⁴	wctype.h	int iswspace (wint_t wc);	Checks for any wide character that corresponds to an implementation-defined set of wide characters for which iswalnum is false.
iswupper ⁴	wctype.h	int iswupper (wint_t wc);	Checks for any uppercase wide character.
iswxdigit ⁴	wctype.h	int iswxdigit (wint_t wc);	Checks for any hexadecimal digit character.
isxdigit ⁴	wctype.h	int isxdigit(int c);	Tests if <i>c</i> is a hexadecimal digit.
j0	math.h	double j0(double x);	Calculates the Bessel function value of the first kind of order 0.
j1	math.h	double j1(double <i>x</i>);	Calculates the Bessel function value of the first kind of order 1.
jn	math.h	double jn(int n , double x);	Calculates the Bessel function value of the first kind of order <i>n</i> .
labs	stdlib.h	long int labs(long int n);	Calculates the absolute value of <i>n</i> .
ldexp	math.h	double ldexp(double <i>x</i> , int <i>exp</i>);	Returns the value of x multiplied by (2 to the power of exp).
ldiv	stdlib.h	ldiv_t ldiv(long int numerator, long int denominator);	Calculates the quotient and remainder of numerator/denominator.
localeconv	locale.h	struct lconv *localeconv(void);	Formats numeric quantities in struct lconv according to the current locale.
localtime	time.h	struct tm *localtime(const time_t *timeval);	Converts <i>timeval</i> to a structure of type tm.

Table 32. Standard C Library Functions (continued)

Function	System Include File	Function Prototype	Description
localtime_r	time.h	struct tm *localtime_r (const time_t *timeval, struct tm *result);	Converts a <i>timer</i> value to a structure of type <i>tm</i> . (Restartable version of localtime.)
log	math.h	double log(double <i>x</i>);	Calculates the natural logarithm of <i>x</i> .
log10	math.h	double log10(double <i>x</i>);	Calculates the base 10 logarithm of <i>x</i> .
longjmp	setjmp.h	void longjmp(jmp_buf env, int value);	Restores a stack environment previously set in <i>env</i> by the setjmp function.
malloc	stdlib.h	void *malloc(size_t size);	Reserves a block of storage.
mblen	stdlib.h	<pre>int mblen(const char *string, size_t n);</pre>	Determines the length of a multibyte character <i>string</i> .
mbrlen ⁴	wchar.h	<pre>int mbrlen (const char *s, size_t n, mbstate_t *ps);</pre>	Determines the length of a multibyte character. (Restartable version of mblen.)
mbrtowc ⁴	wchar.h	int mbrtowc (wchar_t *pwc, const char *s, size_t n, mbstate_t *ps);	Convert a multibyte character to a wide character (Restartable version of mbtowc.)
mbsinit ⁴	wchar.h	int mbsinit (const mbstate_t *ps);	Test state object *ps for initial state.
mbsrtowcs ⁴	wchar.h	size_t mbsrtowc (wchar_t *dst, const char **src, size_t len, mbstate_t *ps);	Convert multibyte string to a wide character string. (Restartable version of mbstowcs.)
mbstowcs	stdlib.h	size_t mbstowcs(wchar_t *pwc, const char *string, size_t n);	Converts the multibyte characters in <i>string</i> to their corresponding wchar_t codes, and stores not more than <i>n</i> codes in <i>pwc</i> .
mbtowc	stdlib.h	<pre>int mbtowc(wchar_t *pwc, const char *string, size_t n);</pre>	Stores the wchar_t code corresponding to the first <i>n</i> bytes of multibyte character <i>string</i> into the wchar_t character <i>pwc</i> .
memchr	string.h	<pre>void *memchr(const void *buf, int c, size_t count);</pre>	Searches the first <i>count</i> bytes of <i>buf</i> for the first occurrence of <i>c</i> converted to an unsigned character.
memcmp	string.h	<pre>int memcmp(const void *buf1, const void *buf2, size_t count);</pre>	Compares up to <i>count</i> bytes of <i>buf1</i> and <i>buf2</i> .
memcpy	string.h	<pre>void *memcpy(void *dest, const void *src, size_t count);</pre>	Copies count bytes of src to dest.
memmove	string.h	void *memmove(void *dest, const void *src, size_t count);	Copies <i>count</i> bytes of <i>src</i> to <i>dest</i> . Allows copying between objects that overlap.

Table 32. Standard C Library Functions (continued)

Function	System Include File	Function Prototype	Description
memset	string.h	<pre>void *memset(void *dest, int c, size_t count);</pre>	Sets <i>count</i> bytes of <i>dest</i> to a value <i>c</i> .
mktime	time.h	time_t mktime(struct tm *time);	Converts local <i>time</i> into calendar time.
modf	math.h	double modf(double <i>x</i> , double * <i>intptr</i>);	Breaks down the floating-point value <i>x</i> into fractional and integral parts.
nl_langinfo ⁴	langinfo.h	char *nl_langinfo(nl_item item);	Retrieve from the current locale the string that describes the requested information specified by <i>item</i> .
perror	stdio.h	<pre>void perror(const char *string);</pre>	Prints an error message to stderr.
pow	math.h	double pow(double <i>x</i> , double <i>y</i>);	Calculates the value <i>x</i> to the power <i>y</i> .
printf	stdio.h	<pre>int printf(const char *format-string, arg-list);</pre>	Formats and prints characters and values to stdout.
putc ¹	stdio.h	int putc(int c, FILE *stream);	Prints <i>c</i> to the output <i>stream</i> .
putchar ¹	stdio.h	int putchar(int c);	Prints <i>c</i> to stdout.
putenv	stdlib.h	int *putenv(const char *varname);	Sets the value of an environment variable by altering an existing variable or creating a new one.
puts	stdio.h	int puts(const char *string);	Prints a string to stdout.
putwc ⁶	stdio.h wchar.h	wint_t putwchar(wchar_t wc, FILE *stream);	Converts the wide character <i>wc</i> to a multibyte character, and writes it to the stream at the current position.
putwchar ⁶	wchar.h	wint_t putwchar(wchar_t wc);	Converts the wide character <i>wc</i> to a multibyte character and writes it to stdout.
qsort	stdlib.h	<pre>void qsort(void *base, size_t num, size_t width, int(*compare)(const void *element1, const void *element2));</pre>	Performs a quick sort of an array of <i>num</i> elements, each of <i>width</i> bytes in size.
raise	signal.h	int raise(int sig);	Sends the signal <i>sig</i> to the running program.
rand	stdlib.h	int rand(void);	Returns a pseudo-random integer.
rand_r	stdlib.h	int rand_r(void);	Returns a pseudo-random integer. (Restartable version)
realloc	stdlib.h	void *realloc(void *ptr, size_t size);	Changes the <i>size</i> of a previously reserved storage block.

Table 32. Standard C Library Functions (continued)

Function	System Include File	Function Prototype	Description
regcomp	regex.h	int regcomp(regex_t *preg, const char *pattern, int cflags);	Compiles the source regular expression pointed to by <i>pattern</i> into an executable version and stores it in the location pointed to by <i>preg</i> .
regerror	regex.h	size_t regerror(int errcode, const regex_t *preg, char *errbuf, size_t errbuf_size);	Finds the description for the error code <i>errcode</i> for the regular expression <i>preg</i> .
regexec	regex.h	<pre>int regexec(const regex_t *preg, const char *string, size_t nmatch, regmatch_t *pmatch, int eflags);</pre>	Compares the null-ended string <i>string</i> against the compiled regular expression <i>preg</i> to find a match between the two.
regfree	regex.h	void regfree(regex_t *preg);	Frees any memory that was allocated by regcomp to implement the regular expression <i>preg</i> .
remove	stdio.h	int remove(const char *filename);	Deletes the file specified by <i>filename</i> .
rename	stdio.h	int rename(const char *oldname, const char *newname);	Renames the specified file.
rewind ¹	stdio.h	void rewind(FILE *stream);	Repositions the file pointer associated with <i>stream</i> to the beginning of the file.
scanf	stdio.h	int scanf(const char *format-string, arg-list);	Reads data from stdin into locations given by <i>arg-list</i> .
setbuf	stdio.h	void setbuf(FILE *stream, char *buffer);	Controls buffering for stream.
setjmp	setjmp.h	int setjmp(jmp_buf env);	Saves a stack environment that can be subsequently restored by longjmp.
setlocale	locale.h	char *setlocale(int category, const char *locale);	Changes or queries variables defined in the <i>locale</i> .
setvbuf	stdio.h	int setvbuf(FILE *stream, char *buf, int type, size_t size);	Controls buffering and buffer size for stream.
signal	signal.h	void(*signal (int sig, void(*func)(int))) (int);	Registers func as a signal handler for the signal sig.
sin	math.h	double sin(double <i>x</i>);	Calculates the sine of x .
sinh	math.h	double sinh(double <i>x</i>);	Calculates the hyperbolic sine of <i>x</i> .
snprintf	stdio.h	int snprintf(char *outbuf, size_t n, const char*,)	Same as sprintf except that the function will stop after n characters have been written to outbuf.

Table 32. Standard C Library Functions (continued)

Function	System Include File	Function Prototype	Description
sprintf	stdio.h	<pre>int sprintf(char *buffer, const char *format-string, arg-list);</pre>	Formats and stores characters and values in <i>buffer</i> .
sqrt	math.h	double sqrt(double x);	Calculates the square root of <i>x</i> .
srand	stdlib.h	void srand(unsigned int seed);	Sets the <i>seed</i> for the pseudo-random number generator.
sscanf	stdio.h	<pre>int sscanf(const char *buffer, const char *format, arg-list);</pre>	Reads data from <i>buffer</i> into the locations given by <i>arg-list</i> .
strcasecmp	strings.h	int srtcasecmp(const char *string1, const char *string2);	Compares strings without case sensitivity.
strcat	string.h	char *strcat(char *string1, const char *string2);	Concatenates string2 to string1.
strchr	string.h	char *strchr(const char *string, int c);	Locates the first occurrence of <i>c</i> in <i>string</i> .
strcmp	string.h	<pre>int strcmp(const char *string1, const char *string2);</pre>	Compares the value of <i>string1</i> to <i>string2</i> .
strcoll	string.h	<pre>int strcoll(const char *string1, const char *string2);</pre>	Compares two strings using the collating sequence in the current locale.
strcpy	string.h	char *strcpy(char *string1, const char *string2);	Copies string2 into string1.
strcspn	string.h	<pre>size_t strcspn(const char *string1, const char *string2);</pre>	Returns the length of the initial substring of <i>string1</i> consisting of characters not contained in <i>string2</i> .
strerror	string.h	char *strerror(int errnum);	Maps the error number in <i>errnum</i> to an error message string.
strfmon ⁴	wchar.h	int strfmon (char *s, size_t maxsize, const char *format,);	Converts monetary value to string.
strftime	time.h	size_t strftime (char *dest, size_t maxsize, const char *format, const struct tm *timeptr);	Stores characters in an array pointed to by <i>dest</i> , according to the string determined by <i>format</i> .
strlen	string.h	size_t strlen(const char *string);	Calculates the length of <i>string</i> .
strncasecmp	strings.h	int srtncasecmp(const char *string1, const char *string2, size_t count);	Compares strings without case sensitivity.

Table 32. Standard C Library Functions (continued)

Function	System Include File	Function Prototype	Description
strncat	string.h	char *strncat(char *string1, const char *string2, size_t count);	Concatenates up to <i>count</i> characters of <i>string2</i> to <i>string1</i> .
strncmp	string.h	int strncmp(const char *string1, const char *string2, size_t count);	Compares up to <i>count</i> characters of <i>string1</i> and <i>string2</i> .
strncpy	string.h	char *strncpy(char *string1, const char *string2, size_t count);	Copies up to <i>count</i> characters of <i>string2</i> to <i>string1</i> .
strpbrk	string.h	char *strpbrk(const char *string1, const char *string2);	Locates the first occurrence in <i>string1</i> of any character in <i>string2</i> .
strptime ⁴	time.h	char *strptime (const char *buf, const char *format, struct tm *tm);	Date and time conversion
strrchr	string.h	char *strrchr(const char *string, int c);	Locates the last occurrence of <i>c</i> in <i>string</i> .
strspn	string.h	size_t strspn(const char *string1, const char *string2);	Returns the length of the initial substring of <i>string1</i> consisting of characters contained in <i>string2</i> .
strstr	string.h	char *strstr(const char *string1, const char *string2);	Returns a pointer to the first occurrence of <i>string2</i> in <i>string1</i> .
strtod	stdlib.h	double strtod(const char *nptr, char **endptr);	Converts <i>nptr</i> to a double precision value.
strtok	string.h	char *strtok(char *string1, const char *string2);	Locates the next token in <i>string1</i> delimited by the next character in <i>string2</i> .
strtok_r	string.h	char *strtok_r(char *string, const char *seps, char **lasts);	Locates the next token in <i>string</i> delimited by the next character in <i>seps</i> . (Restartable version of strtok.)
strtol	stdlib.h	long int strtol(const char *nptr, char **endptr, int base);	Converts <i>nptr</i> to a signed long integer.
strtoul	stdlib.h	unsigned long int strtoul(const char *string1, char **string2, int base);	Converts <i>string1</i> to an unsigned long integer.
strxfrm	string.h	size_t strxfrm(char *string1, const char *string2, size_t count);	Converts <i>string2</i> and places the result in <i>string1</i> . The conversion is determined by the program's current locale.
swprintf	wchar.h	<pre>int swprintf(wchar_t *wcsbuffer, size_t n, const wchar_t *format, arg-list);</pre>	Formats and stores a series of wide characters and values into the wide-character buffer <i>wcsbuffer</i> .

Table 32. Standard C Library Functions (continued)

Function	System Include File	Function Prototype	Description
swscanf	wchar.h	int swscanf (const wchar_t *buffer, const wchar_t *format, arg-list)	Reads data from <i>buffer</i> into the locations given by <i>arg-list</i> .
system	stdlib.h	<pre>int system(const char *string);</pre>	Passes <i>string</i> to the system command analyzer.
tan	math.h	double tan(double x);	Calculates the tangent of <i>x</i> .
tanh	math.h	double tanh(double x);	Calculates the hyperbolic tangent of <i>x</i> .
time	time.h	time_t time(time_t *timeptr);	Returns the current calendar time.
tmpfile	stdio.h	FILE *tmpfile(void);	Creates a temporary binary file and opens it.
tmpnam	stdio.h	char *tmpnam(char *string);	Generates a temporary file name.
toascii	ctype.h	int toascii(int c);	Converts <i>c</i> to a character in the 7-bit US-ASCII character set.
tolower	ctype.h	int tolower(int c);	Converts c to lowercase.
toupper	ctype.h	int toupper(int c);	Converts <i>c</i> to uppercase.
towctrans	wctype.h	wint_t towctrans(wint_t wc, wctrans_t desc);	Translates the wide character <i>wc</i> based on the mapping described by <i>desc</i> .
towlower ⁴	wctype.h	wint_t towlower (wint_t wc);	Converts uppercase letter to lowercase letter.
towupper ⁴	wctype.h	wint_t towupper (wint_t wc);	Converts lowercase letter to uppercase letter.
ungetc ¹	stdio.h	int ungetc(int c, FILE *stream);	Pushes <i>c</i> back onto the input <i>stream</i> .
ungetwc ⁶	stdio.h wchar.h	wint_t ungetwc(wint_t wc, FILE *stream);	Pushes the wide character <i>wc</i> back onto the input stream.
va_arg	stdarg.h	<pre>var_type va_arg(va_list arg_ptr, var_type);</pre>	Returns the value of one argument and modifies <i>arg_ptr</i> to point to the next argument.
va_end	stdarg.h	void va_end(va_list arg_ptr);	Facilitates normal return from variable argument list processing.
va_start	stdarg.h	void va_start(va_list arg_ptr, variable_name);	Initializes arg_ptr for subsequent use by va_arg and va_end.
vfprintf	stdio.h stdarg.h	int vfprintf(FILE *stream, const char *format, va_list arg_ptr);	Formats and prints characters to the output <i>stream</i> using a variable number of arguments.
vfwprintf ⁶	stdarg.h stdio.h wchar.h	<pre>int vfwprintf(FILE *stream, const wchar_t *format, va_list arg);</pre>	Equivalent to fwprintf, except that the variable argument list is replaced by <i>arg</i> .

Table 32. Standard C Library Functions (continued)

Function	System Include File	Function Prototype	Description
vprintf	stdio.h stdarg.h	<pre>int vprintf(const char *format, va_list arg_ptr);</pre>	Formats and prints characters to stdout using a variable number of arguments.
vsprintf	stdio.h stdarg.h	int vsprintf(char *target-string, const char *format, va_list arg_ptr);	Formats and stores characters in a buffer using a variable number of arguments.
vsnprintf	stdio.h	int vsnprintf(char *outbuf, size_t n, const char*, va_list);	Same as vsprintf except that the function will stop after n characters have been written to outbuf.
vswprintf	stdarg.h wchar.h	<pre>int vswprintf(wchar_t *wcsbuffer, size_t n, const wchar_t *format, va_list arg);</pre>	Formats and stores a series of wide characters and values in the buffer <i>wcsbuffer</i> .
vwprintf ⁶	stdarg.h wchar.h	<pre>int vwprintf(const wchar_t *format, va_list arg);</pre>	Equivalent to wprintf, except that the variable arument list is replaced by <i>arg</i> .
wcrtomb ⁴	wchar.h	int wcrtomb (char *s, wchar_t wchar, mbstate_t *pss);	Converts a wide character to a multibyte character. (Restartable version of wctomb.)
wcscat	wcstr.h	<pre>wchar_t *wcscat(wchar_t *string1, const wchar_t *string2);</pre>	Appends a copy of the string pointed to by <i>string2</i> to the end of the string pointed to by <i>string1</i> .
wcschr	wcstr.h	wchar_t *wcschr(const wchar_t *string, wchar_t character);	Searches the wide-character string pointed to by <i>string</i> for the occurrence of <i>character</i> .
wcscmp	wcstr.h	int wcscmp(const wchar_t *string1, const wchar_t *string2);	Compares two wide-character strings, *string1 and *string2.
wcscoll ⁴	wchar.h	int wcscoll (const wchar_t *wcs1, const wchar_t *wcs2);	Compares two wide-character strings using the collating sequence in the current locale.
wcscpy	wcstr.h	<pre>wchar_t *wcscpy(wchar_t *string1, const wchar_t *string2);</pre>	Copies the contents of *string2 (including the ending wchar_t null character) into *string1.
wcscspn	wcstr.h	size_t wcscspn(const wchar_t *string1, const wchar_t *string2);	Determines the number of wchar_t characters in the initial segment of the string pointed to by *string1 that do not appear in the string pointed to by *string2.
wcsftime	wchar.h	size_t wcsftime(wchar_t *wdest, size_t maxsize, const wchar_t *format, const struct tm *timeptr);	Converts the time and date specification in the <i>timeptr</i> structure into a wide-character string.
wcslen	wcstr.h	size_t wcslen(const wchar_t *string);	Computes the number of wide-characters in the string pointed to by <i>string</i> .

Table 32. Standard C Library Functions (continued)

Function	System Include File	Function Prototype	Description
wcsncat	wcstr.h	<pre>wchar_t *wcsncat(wchar_t *string1, const wchar_t *string2, size_t count);</pre>	Appends up to <i>count</i> wide characters from <i>string2</i> to the end of <i>string1</i> , and appends a wchar_t null character to the result.
wesnemp	wcstr.h	<pre>int wcsncmp(const wchar_t *string1, const wchar_t *string2, size_t count);</pre>	Compares up to <i>count</i> wide characters in <i>string1</i> to <i>string2</i> .
wcsncpy	wcstr.h	<pre>wchar_t *wcsncpy(wchar_t *string1, const wchar_t *string2, size_t count);</pre>	Copies up to <i>count</i> wide characters from <i>string2</i> to <i>string1</i> .
wcspbrk	wcstr.h	wchar_t *wcspbrk(const wchar_t *string1, const wchar_t *string2);	Locates the first occurrence in the string pointed to by <i>string1</i> of any wide characters from the string pointed to by <i>string2</i> .
wcsrchr	wcstr.h	wchar_t *wcsrchr(const wchar_t *string, wchar_t character);	Locates the last occurrence of <i>character</i> in the string pointed to by <i>string</i> .
wcsrtombs ⁴	wchar.h	size_t wcsrtombs (char *dst, const wchar_t **src, size_t len, mbstate_t *ps);	Converts wide character string to multibyte string. (Restartable version of wcstombs.)
wcsspn	wchar.h	size_t wcsspn(const wchar_t *string1, const wchar_t *string2);	Computes the number of wide characters in the initial segment of the string pointed to by <i>string1</i> , which consists entirely of wide characters from the string pointed to by <i>string2</i> .
wcsstr	wchar.h	wchar_t *wcsstr(const wchar_t *wcs1, const wchar_t *wcs2);	Locates the first occurrence of wcs2 in wcs1.
wcstod	wchar.h	double wcstod(const wchar_t *nptr, wchar_t **endptr);	Converts the initial portion of the wide-character string pointed to by <i>nptr</i> to a double value.
wcstok	wchar.h	wchar_t *wcstok(wchar_t *wcs1, const wchar_t *wcs2, wchar_t **ptr)	Breaks <i>wcs1</i> into a sequence of tokens, each of which is delimited by a wide character from the wide string pointed to by <i>wcs2</i> .
wcstol	wchar.h	long int wcstol(const wchar_t *nptr, wchar_t **endptr, int base);	Converts the initial portion of the wide-character string pointed to by <i>nptr</i> to a long integer value.
wcstombs	stdlib.h	size_t wcstombs(char *dest, const wchar_t *string, size_t count);	Converts the wchar_t <i>string</i> into a multibyte string <i>dest</i> .

Table 32. Standard C Library Functions (continued)

Function	System Include File	Function Prototype	Description
wcstoul	wchar.h	unsigned long int wcstoul(const wchar_t *nptr, wchar_t **endptr, int base);	Converts the initial portion of the wide-character string pointed to by <i>nptr</i> to an unsigned long integer value.
wcsxfrm ⁴	wchar.h	size_t wcsxfrm (wchar_t *wcs1, const wchar_t *wcs2, size_t n);	Transforms a wide-character string to values which represent character collating weights and places the resulting wide-character string into an array.
wctob	stdarg.h wchar.h	int wctob(wint_t wc);	Determines whether <i>wc</i> corresponds to a member of the extended character set whose multibyte character representation is a single byte when in the initial shift state.
wctomb	stdlib.h	<pre>int wctomb(char *string, wchar_t character);</pre>	Converts the wchar_t value of <i>character</i> into a multibyte <i>string</i> .
wctrans	wctype.h	wctrans_t wctrans(const char *property);	Constructs a value with type wctrans_t that describes a mapping between wide characters identified by the string argument property.
wctype ⁴	wchar.h	wctype_t wctype (const char *property);	Obtains handle for character property classification.
wcwidth	wchar.h	<pre>int wcswidth(const wchar_t *pwcs, size_t n);</pre>	Determine the display width of a wide character string.
wmemchr	wchar.h	<pre>wchar_t *wmemchr(const wchar_t *s, wchar_t c, size_t n);</pre>	Locates the first occurrence of <i>c</i> in the initial <i>n</i> wide characters of the object pointed to by <i>s</i> .
wmemcmp	wchar.h	int wmemcmp(const wchar_t *s1, const wchar_t *s2, size_t n);	Compares the first <i>n</i> wide characters of the object pointed to by <i>s</i> 1 to the first <i>n</i> characters of the object pointed to by <i>s</i> 2.
wmemcpy	wchar.h	<pre>wchar_t *wmemcpy(wchar_t *s1, const wchar_t *s2, size_t n);</pre>	Copies <i>n</i> wide characters from the object pointed to by <i>s</i> 2 to the object pointed to by <i>s</i> 1.
wmemmove	wchar.h	<pre>wchar_t *wmemmove(wchar_t *s1, const wchar_t *s2, size_t n);</pre>	Copies <i>n</i> wide characters from the object pointed to by <i>s</i> 2 to the object pointed to by <i>s</i> 1.
wmemset	wchar.h	<pre>wchar_t *wmemset(wchar_t *s, wchar_t c, size_t n);</pre>	Copies the value of <i>c</i> into each of the first <i>n</i> wide chracters of the object pointed to by <i>s</i> .

Table 32. Standard C Library Functions (continued)

Function	System Include File	Function Prototype	Description
wprintf ⁶	wchar.h	<pre>int wprintf(const wchar_t *format, arg-list);</pre>	Equivalent to fwprintf with the argument stdout interposed before the arguments to wprintf.
wscanf ⁶	wchar.h	<pre>int wscanf(const wchar_t *format, arg-list);</pre>	Equivalent to fwscanf with the argument stdin interposed before the arguments of wscanf.
y0	math.h	double y0(double x);	Calculates the Bessel function value of the second kind of order 0.
y1	math.h	double y1(double x);	Calculates the Bessel function value of the second kind of order 1.
yn	math.h	double yn(int <i>n</i> , double <i>x</i>);	Calculates the Bessel function value of the second kind of order <i>n</i> .

ILE C Library Extensions to C Library Functions Table

This table briefly describes all the ILE C library extensions, listed in alphabetical order. This table provides the include file name, and the function prototype for each function.

Table 33. ILE C Library Extensions

Function	System Include file	Function prototype	Description
_C_Get _Ssn_Handle	stdio.h	_SSN_Handle_T _C_Get_Ssn_Handle (void);	Returns a handle to the C session for use with DSM APIs.
_C_TS _malloc64	stdlib.h	void *_C_TS_malloc64(unsigned long long int);	Same as _C_TS_malloc, but takes an unsigned long long int so the user can ask for more than 2 GB of storage on a single request.
_C_TS _malloc_info	mallocinfo.h	void *malloc(size_t size);	Returns current memory usage information.
_C_TS _malloc_debug	mallocinfo.h	void *malloc(size_t size);	Returns the same information as _C_TS_malloc_info, plus produces a spool file of detailed information about the memory structure used by C_TS_malloc functions.

Table 33. ILE C Library Extensions (continued)

Function	System Include file	Function prototype	Description
_GetExcData	signal.h	void _GetExcData (_INTRPT_Hndlr_Parms_T *parms);	Retrieves information about an exception from within a signal handler.
QXXCHGDA	xxdtaa.h	void QXXCHGDA(_DTAA_NAME_ dtaname, short int offset, short int len, char *dtaptr);	Changes the OS/400 Thata area specified on <i>dtaname</i> using the data pointed to by <i>dtaptr</i> .
QXXDTOP	xxcvt.h	void QXXDTOP(unsigned char *pptr, int digits, int fraction, double value);	Converts a double value to a packed decimal value with digits total digits and fraction fractional digits.
QXXDTOZ	xxcvt.h	void QXXDTOZ(unsigned char *zptr, int digits, int fraction, double value);	Converts a double value to a zoned decimal value with digits total digits and fraction fractional digits.
QXXITOP	xxcvt.h	void QXXITOP(unsigned char *pptr, int digits, int fraction, int value);	Converts an integer value to a packed decimal value.
QXXITOZ	xxcvt.h	void QXXITOZ(unsigned char *zptr, int digits, int fraction, int value);	Converts an integer value to a zoned decimal value.
QXXPTOD	xxcvt.h	double QXXPTOD(unsigned char *pptr, int digits, int fraction);	Converts a packed decimal number to a double value with digits total digits and fraction fractional digits.
QXXPTOI	xxcvt.h	int QXXPTOI(unsigned char *pptr, int digits, int fraction);	Converts a packed decimal number to an integer value with digits total digits and fraction fractional digits.
QXXRTVDA	xxdtaa.h	void QXXRTVDA(_DTAA_NAME_' dtaname, short int offset, short int len, char *dtaptr);	Retrieves a copy of Tthe OS/400 data area specified on <i>dtaname</i> .
QXXZTOD	xxcvt.h	double QXXZTOD(unsigned char *zptr, int digits, int fraction);	Converts a zoned decimal number to a double value with digits total digits and fraction fractional digits.

Table 33. ILE C Library Extensions (continued)

Function	System Include file	Function prototype	Description
QXXZTOI	xxcvt.h	int QXXZTOI(unsigned char *zptr, int digits, int fraction);	Converts a zoned decimal value to an integer value with digits total digits and fraction fractional digits.
_Racquire	recio.h	int _Racquire(_RFILE *fp, char *dev);	Prepares a device for record I/O operations.
_Rclose	recio.h	int _Rclose(_RFILE *fp);	Closes a file that is opened for record I/O operations.
_Rcommit	recio.h	int _Rcommit(char *cmtid);	Completes the current transaction, and establishes a new commitment boundary.
_Rdelete	recio.h	_RIOFB_T *_Rdelete(_RFILE *fp);	Deletes the currently locked record.
_Rdevatr	xxfdbk.h recio.h	_XXDEV_ATR_T *_Rdevatr(_RFILE *fp, char *pgmdev);	Returns a pointer to a copy of the device attributes feedback area for the file referenced by <i>fp</i> and the device <i>pgmdev</i> .
_Rfeod	recio.h	int _Rfeod(_RFILE *fp);	Forces an end-of-file condition for the file referenced by <i>fp</i> .
_Rfeov	recio.h	int _Rfeov(_RFILE *fp);	Forces an end-of-volume condition for the tape file referenced by fp.
_Rformat	recio.h	void Rformat(_RFILE *fp, char *fmt);	Sets the record format to <i>fmt</i> for the file referenced by <i>fp</i> .
_Rindara	recio.h	void _Rindara (_RFILE *fp, char *indic_buf);	Sets up the separate indicator area to be used for subsequent record I/O operations.
_Riofbk	recio.h xxfdbk.h	_XXIOFB_T *_Riofbk(_RFILE *fp);	Returns a pointer to a copy of the I/O feedback area for the file referenced by fp.
_Rlocate	recio.h	_RIOFB_T *_Rlocate(_RFILE *fp, void *key, int klen_rrn, int opts);	Positions to the record in the file associated with <i>fp</i> and specified by the <i>key</i> , <i>klen_rrn</i> and <i>opt parameters</i> .
_Ropen	recio.h	_RFILE *_Ropen(const char *filename, const char *mode);	Opens a file for record I/O operations.

Table 33. ILE C Library Extensions (continued)

Function	System Include file	Function prototype	Description
_Ropnfbk	recio.h xxfdbk.h	_XXOPFB_T *_Ropnfbk(_RFILE *fp);	Returns a pointer to a copy of the open feedback area for the file referenced by fp.
_Rpgmdev	recio.h	int _Rpgmdev(_RFILE *fp, char *dev);	Sets the default program device.
_Rreadd	recio.h	_RIOFB_T *_Rreadd(_RFILE *fp, void *buf, size_t size, int opts, long rrn);	Reads a record by relative record number.
_Rreadf	recio.h	_RIOFB_T *_Rreadf(_RFILE *fp, void *buf, size_t size, int opts);	Reads the first record.
_Rreadindv	recio.h	_RIOFB_T *_Rreadindv(_RFILE *fp, void *buf, size_t size, int opts);	Reads a record from an invited device.
_Rreadk	recio.h	_RIOFB_T *_Rreadk(_RFILE *fp, void *buf, size_t size, int opts, void *key, int klen);	Reads a record by key.
_Rreadl	recio.h	_RIOFB_T *_Rreadl(_RFILE *fp, void *buf, size_t size, int opts);	Reads the last record.
_Rreadn	recio.h	_RIOFB_T *_Rreadn(_RFILE *fp, void *buf, size_t size, int opts);	Reads the next record.
_Rreadnc	recio.h	_RIOFB_T *_Rreadnc(_RFILE *fp, void *buf, size_t size);	Reads the next changed record in the subfile.
_Rreadp	recio.h	_RIOFB_T *_Rreadp(_RFILE *fp, void *buf, size_t size, int opts);	Reads the previous record.
_Rreads	recio.h	_RIOFB_T *_Rreads(_RFILE *fp, void *buf, size_t size, int opts);	Reads the same record.
_Rrelease	recio.h	int _Rrelease(_RFILE *fp, char *dev);	Makes the specified device ineligible for record I/O operations.
_Rrlslck	recio.h	int _Rrlslck(_RFILE *fp);	Releases the currently locked record.
_Rrollbck	recio.h	int _Rrollbck(void);	Reestablishes the last commitment boundary as the current commitment boundary.
_Rupdate	recio.h	_RIOFB_T *_Rupdate(_RFILE *fp, void *buf, size_t size);	Writes to the record that is currently locked for update.

Table 33. ILE C Library Extensions (continued)

Function	System Include file	Function prototype	Description
_Rupfb	recio.h	_RIOFB_T *_Rupfb(_RFILE *fp);	Updates the feedback structure with information about the last record I/O operation.
_Rwrite	recio.h	_RIOFB_T *_Rwrite(_RFILE *fp, void *buf, size_t size);	Writes a record to the end of the file.
_Rwrited	recio.h	_RIOFB_T *_Rwrited(_RFILE *fp, void *buf, size_t size, unsigned long rrn);	Writes a record by relative record number. It only writes over deleted records.
_Rwriterd	recio.h	_RIOFB_T *_Rwriterd(_RFILE *fp, void *buf, size_t size);	Reads and writes a record.
_Rwrread	recio.h	_RIOFB_T *_Rwrread(_RFILE *fp, void *inbuf, size_t in_buf_size, void *out_buf, size_t out_buf_size);	Functions as _Rwriterd, except separate buffers may be specified for input and output data.
wcsicmp	wchar.h	<pre>intwcsicmp(const wchar_t *string1, const wchar_t *string2);</pre>	Compares wide character strings without case sensitivity.
wcsnicmp	wchar.h	<pre>intwcsnicmp(const wchar_t *string1, const wchar_t *string2, size_t count);</pre>	Compares wide character strings without case sensitivity.

Appendix B. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing IBM Corporation 500 Columbus Avenue Thornwood, NY 10594 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing 2-31 Roppongi 3-chome, Minato-ku Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created

programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation Software Interoperability Coordinator 3605 Highway 52 N Rochester, MN 55901-7829 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy,

modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This publication is intended to help you to write Integrated Language Environment C/C++ for OS/400 programs. It contains information necessary to use the Integrated Language Environment C/C++ for the OS/400 compiler. The ILE C/C++ Programmer's Guide primarily documents general-use programming interfaces and associated guidance information provided by the Integrated Language Environment C/C++ for the OS/400 compiler.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

Application System/400 AS/400 e (logo) **IBM** *iSeries* Operating System/400 OS/400 400

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

PC Direct is a trademark of Ziff Communications Company in the United States, other countries, or both and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium, and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET and the SET Logo are trademarks owned by SET Secure Electronic Transaction

Other company, product, and service names may be trademarks or service marks

Bibliography

For additional information about topics related to ILE C programming on the iSeries system, refer to the following IBM iSeries publications:

- WebSphere Development Studio: Application
 Development Manager User's Guide,
 SC09-2133-02, describes creating and managing
 projects defined for the Application
 Development Manager/400 feature, as well as
 using the program to develop applications.
- ADTS/400: Programming Development Manager, SC09-1771-00, provides information about using the Application Development ToolSet/400 programming development manager (PDM) to work with lists of libraries, objects, members, and user-defined options to easily do such operations as copy, delete, and rename.
 Contains activities and reference material to help the user learn PDM. The most commonly used operations and function keys are explained in detail using examples.
- ADTS for AS/400: Source Entry Utility,
 SC09-2605-00, provides information about using
 the Application Development ToolSet/400
 source entry utility (SEU) to create and edit
 source members. The manual explains how to
 start and end an SEU session and how to use
 the many features of this full-screen text editor.
 The manual contains examples to help both
 new and experienced users accomplish various
 editing tasks, from the simplest line commands
 to using pre-defined prompts for high-level
 languages and data formats.
- Application Display Programming, SC41-5715-00, provides information about:
 - Using DDS to create and maintain displays for applications;
 - Creating and working with display files on the system;
 - Creating online help information;
 - Using UIM to define panels and dialogs for an application;
 - Using panel groups, records, or documents
- The Backup and Recovery Information Center topic provides information about setting up and managing the following:
 - Journaling, access path protection, and commitment control
 - User auxiliary storage pools (ASPs)

Disk protection (device parity, mirrored, and checksum)

Provides performance information about backup media and save/restore operations. Also includes advanced backup and recovery topics, such as using save-while-active support, saving and restoring to a different release, and programming tips and techniques.

- CICS for iSeries Application Programming Guide, SC41-5454-01, provides information on application programming for CICS/400[®]. It includes guidance and reference information on the CICS application programming interface and system programming interface commands, and gives general information about developing new applications and migrating existing applications from other CICS platforms.
- *ILE C/C++ for AS/400 MI Library Reference*, SC09-2418-00, provides information on Machine Interface instructions available in the ILE C compiler that provide system-level programming capabilities.
- CL Programming, SC41-5721-05, provides a
 wide-ranging discussion of iSeriesprogramming
 topics including a general discussion on objects
 and libraries, CL programming, controlling
 flow and communicating between programs,
 working with objects in CL programs, and
 creating CL programs. Other topics include
 predefined and impromptu messages and
 message handling, defining and creating
 user-defined commands and menus, application
 testing, including debug mode, breakpoints,
 traces, and display functions.
- The CL Reference topic provides a description of the iSeriescontrol language (CL) and its OS/400 commands. (Non-OS/400 commands are described in the respective licensed program publications.) Also provides an overview of *all* the CL commands for the iSeries system, and it describes the syntax rules needed to code them.
- P Communications Management, SC41-5406-02, provides information about work management in a communications environment, communications status, tracing and diagnosing communications problems, error handling and

- recovery, performance, and specific line speed and subsystem storage information.
- The File Management Information Center topic provides information about using files in application programs. Includes information on the following:
 - Fundamental structure and concepts of data management support on the system
 - Overrides and file redirection (temporarily making changes to files when an application program is run)
 - Copying files by using system commands to copy data from one place to another
 - Tailoring a system using double-byte data
- The Database Programming topic provides a detailed discussion of the iSeries database organization, including information on how to create, describe, and update database files on the system. Also describes how to define files to the system using OS/400 data description specifications (DDS) keywords.
- The SQL Programming topic provides information about how to use DB2[®] Query Manager and SQL Development kit licensed program. Shows how to access data in a database library and prepare, run, and test an application program that contains embedded SQL statements. Contains examples of SQL/400[®] statements and a description of the interactive SQL function. Describes common concepts and rules for using SQL/400 statements in ILE COBOL, PL/I, ILE C, FORTRAN/400[®], ILE RPG/400, and REXX.
- The SQL Reference topic provides information about how to use Structured Query Language/400 DB2 statements and gives details about the proper use of the statements. Examples of statements include syntax diagrams, parameters, and definitions. A list of SQL limits and a description of the SQL communication area (SQLCA) and SQL descriptor area (SQLDA) are also provided.
- The DDS Reference topic provides detailed descriptions for coding the data description specifications (DDS) for file that can be described externally. These files are physical, logical, display, print, and intersystem communication function (ICF) files.
- The Distributed Data Management topic provides information about remote file processing. It describes how to define a remote file to OS/400 distributed data management (DDM), how to create a DDM file, what file

- utilities are supported through DDM, and the requirements of OS/400 DDM as related to other systems.
- Experience RPG IV Multimedia Tutorial, SK2T-2700, is an interactive self-study program explaining the differences between RPG III and RPG IV and how to work within the new ILE environment. An accompanying workbook provides additional exercises and doubles as a reference upon completion of the tutorial. ILE RPG code examples are shipped with the tutorial and run directly on the iSeries.
- GDDM Programming Guide, SC41-0536-00, provides information about using OS/400graphical data display manager (GDDM®) to write graphics application programs. Includes many example programs and information to help users understand how the product fits into data processing systems.
- GDDM Reference, SC41-3718-00, provides information about using OS/400graphical data display manager (GDDM) to write graphics application programs. This manual provides detailed descriptions of all graphics routines available in GDDM. Also provides information about high-level language interfaces to GDDM.
- *ICF Programming*, SC41-5442-00, provides information needed to write application programs that use iSeries communications and the OS/400 intersystem communications function (OS/400-ICF). Also contains information on data description specifications (DDS) keywords, system-supplied formats, return codes, file transfer support, and program examples.
- IDDU Use, SC41-5704-00, describes how to use the iSeries interactive data definition utility (IDDU) to describe data dictionaries, files, and records to the system. Includes:
 - An introduction to computer file and data definition concepts
 - An introduction to the use of IDDU to describe the data used in queries and documents
 - Representative tasks related to creating, maintaining, and using data dictionaries, files, record formats, and fields
 - Advanced information about using IDDU to work with files created on other systems and information about error recovery and problem prevention.
- WebSphere Development Studio: ILE C/C++ Programmer's Guide, SC09-2712-03, provides

- information on how to develop applications using the ILE C language. It includes information about creating, running and debugging programs. It also includes programming considerations for interlanguage program and procedure calls, locales, handling exceptions, database, externally described and device files. Some performance tips are also described. An appendix includes information on migrating source code from EPM C or System C to ILE C.
- ILE C for AS/400 Language Reference, SC09-2711-01, provides reference information about ILE C, including elements of the language, statements, and preprocessor directives. Examples are provided and considerations for programming are also discussed.
- WebSphere Development Studio: ILE COBOL Programmer's Guide, SC09-2540-03, provides information about how to write, compile, bind, run, debug, and maintain ILE COBOL programs on the iSeries system. It provides programming information on how to call other ILE COBOL and non-ILE COBOL programs, share data with other programs, use pointers, and handle exceptions. It also describes how to perform input/output operations on externally attached devices, database files, display files, and ICF files.
- WebSphere Development Studio: ILE COBOL Reference, SC09-2539-03, provides a description of the ILE COBOL programming language. It provides information on the structure of the ILE COBOL programming language and the structure of an ILE COBOL source program. It also provides a description of all Identification Division paragraphs, Environment Division clauses, Data Division clauses, Procedure Division statements, and Compiler-Directing statements.
- WebSphere Development Studio: ILE COBOL Reference Summary, SX09-1317-03, provides quick reference information on the structure of the ILE COBOL programming language and the structure of an ILE COBOL source program. It also provides syntax diagrams of all Identification Division paragraphs, Environment Division clauses, Data Division clauses, Procedure Division statements, and Compiler-Directing statements.
- *ILE Concepts*, SC41-5606-06, explains concepts and terminology pertaining to the Integrated Language Environment architecture of the OS/400 licensed program. Topics covered

- include creating modules, binding, running programs, debugging programs, and handling exceptions.
- WebSphere Development Studio: ILE RPG Programmer's Guide, SC09-2507-04, provides information about the ILE RPG programming language, which is an implementation of the RPG IV language in the Integrated Language Environment (ILE) on the iSeries system. It includes information on creating and running programs, with considerations for procedure calls and interlanguage programming. The guide also covers debugging and exception handling and explains how to use iSeries files and devices in RPG programs. Appendixes include information on migration to RPG IV and sample compiler listings. It is intended for people with a basic understanding of data processing concepts and of the RPG language.
- WebSphere Development Studio: ILE RPG Reference, SC09-2508-04, provides information about the ILE RPG programming language. This manual describes, position by position and keyword by keyword, the valid entries for all RPG IV specifications, and provides a detailed description of all the operation codes and built-in functions. This manual also contains information on the RPG logic cycle, arrays and tables, editing functions, and indicators.
- WebSphere Development Studio: ILE RPG Reference Summary, SX09-1315-03, provides information about the RPG III and RPG IV programming language. This manual contains tables and lists for all specifications and operations in both languages. A key is provided to map RPG III specifications and operations to RPG IV specifications and operations.
- Local Device Configuration, SC41-5121-00, provides information about configuring local devices on the iSeries system. This includes information on how to configure the following:
 - Local work station controllers (including twinaxial controllers)
 - Tape controllers
 - Locally attached devices (including twinaxial devices)
- Machine Interface Functional Reference, SC41-5810-00, describes the machine interface instruction set. Describes the functions that can be performed by each instruction and also the necessary information to code each instruction.
- *Printer Device Programming*, SC41-5713-05, provides information to help you understand and control printing. Provides specific

information on printing elements and concepts of the iSeries system, printer file and print spooling support for printing operations, and printer connectivity. Includes considerations for using personal computers, other printing functions such as Business Graphics Utility (BGU), advanced function printing (AFP), and examples of working with the iSeries system printing elements such as how to move spooled output files from one output queue to a different output queue. Also includes an appendix of control language (CL) commands used to manage printing workload. Fonts available for use with the iSeries system are also provided. Font substitution tables provide a cross-reference of substituted fonts if attached printers do not support application-specified fonts.

- REXX/400 Programmer's Guide, SC41-5728-00, provides a wide-ranging discussion of programming with REXX on the AS/400 system. Its primary purpose is to provide useful programming information and examples to those who are new to Procedures Language 400/REXX and to provide those who have used REXX in other computing environments with information about the Procedures Language 400/REXX implementation.
- WebSphere Development Studio: ILE RPG Programmer's Guide, SC09-2507-04, provides information needed to design, code, compile, and test RPG programs on the AS/400 system. The manual provides information on data structures, data formats, file processing, multiple file processing, the automatic report function, RPG command statements, testing and debugging functions, application design techniques, problem analysis, and compiler service information. The differences between the RPG for iSeries compiler, the System/38® environment RPG III compiler, and the System/36[®]-compatible RPG II compiler are also discussed.
- The Basic Security topic explains why security is necessary, defines major concepts, and provides information on planning, implementing, and monitoring basic security on the iSeries system.
- iSeries Security Reference, SC41-5302-06, tells how system security support can be used to protect the system and the data from being used by people who do not have the proper authorization, protect the data from intentional

- or unintentional damage or destruction, keep security information up-to-date, and set up security on the system.
- Local Device Configuration, SC41-5121-00, provides step-by-step procedures for initial installation, installing licensed programs, program temporary fixes (PTFs), and secondary languages from IBM. This manual is also for users who already have an iSeries system with an installed release and want to install a new
- System API Programming, SC41-5800-00, provides information for the experienced application and system programmers who want to use the OS/400 application programming interfaces (APIs). Provides getting started and examples to help the programmer use APIs.
- The System API Reference topic provides information for the experienced programmer on how to use the application programming interfaces (APIs) to such OS/400 functions as:
 - Dynamic Screen Manager
 - Files (database, spooled, hierarchical)
 - Message handling
 - National language support
 - Network management
 - Objects
 - Problem management
 - Registration facility
 - Security
 - Software products
 - Source debug
 - UNIX-type
 - User-defined communications
 - User interface
 - Work management

Includes original program model (OPM), Integrated Language Environment (ILE), and UNIX-type APIs.

- The Systems Management topic provides information about the system unit control panel, starting and stopping the system, using tapes and diskettes, working with program temporary fixes, as well as handling problems.
- Tape and Diskette Device Programming, SC41-5716-02, provides information to help users develop and support programs that use tape and diskette drives for I/O. Includes information on device files and descriptions for tape and diskette devices.

Index

Special Characters	absolute value (continued)	calculating (continued)
Special Characters	labs 155	floating-point remainder 91
EXBDY built-in 6	access mode 76, 92	hyperbolic cosine 65
VBDY built-in 6	acos() function 39	hyperbolic sine 316
_C_Get_Ssn_Handle() function 55	acquire a program device 228	hypotenuse 145
_C_TS_malloc 171, 490	adding data to streams 76	logarithm 165
_C_TS_malloc_debug 490	append mode	natural logarithm 165
_C_TS_malloc_info 490	using fopen() function 92	quotient and remainder 70
_C_TS_malloc64 171, 490	appending data to files 76	sine 315
_EXBDY macro 6	arccosine 39	time difference 69
_fputchar() function 102	arcsine 43	calloc() function 56
_gcvt() function 132	arctangent 45	cancel handler reason codes 459
_GetExcData() function 135	argument list functions 31	case mapping functions 35
_INTRPT_Hndlr_Parms_T 4	asctime() function 40	catclose() function 57
_itoa() function 154	asctime_r() function 42	catgets() function 58
_ltoa() function 167 _Racquire() function 228	asin() function 43	catopen() function 59
_Rclose() function 229	assert() function 44	ceil() function 61
_Recommit() function 230	assert.h include file 3	ceiling function 61
_Rdelete() function 232	atan() function 45	changing
_Rdevatr() function 233	atan2() function 45	data area 218
_Rfeod() function 246	atexit() function 46	environment variables 211
_Rfeov() function 247	atof() function 47	file position 114
_Rformat() function 249	atoi() function 49	reserved storage block size 234
_Rindara() function 251	atol() function 50 atoll() function	character
_Riofbk() function 252	strings to long long values 50	converting
_Rlocate() function 254	strings to long long values 50	to floating-point 50 to integer 49
_Ropen() function 257		to long integer 50
_Ropnfbk() function 261	В	reading 82, 133
_Rpgmdev() function 262		setting 192
_Rreadd() function 263	bessel functions 23, 51	ungetting 381
_Rreadf() function 265	binary files 94	writing 101, 210
_Rreadindv() function 267	binary search 52	character case mapping
_Rreadk() function 270	blksize 94 block size 94	tolower 377
_Rreadl() function 274	bsearch() function 52	toupper 377
_Rreadn() function 275	btowc() function 54	towlower 379
_Rreadn() function 278	buffers	towupper 379
_Rreadp() function 279 _Rreads() function 282	assigning 305	character testing
_Rrelease() function 283	comparing 188	ASCII value 146
_Rrlslck() function 285	copying 189, 191	character property 149, 152
_Rrollbck() function 286	flushing 80	isalnum 147
_Rupdate() function 288	searching 187	isalpha 147
_Rupfb() function 290	setting characters 192	iscntrl 147
_Rwrite() function 291	bufsiz constant 15	isdigit 147
_Rwrited() function 293	builtins	isgraph 147
_Rwriterd() function 296	_EXBDY 6	islower 147 isprint 147
_Rwrread() function 297	_VBDY 6	ispunct 147
_ultoa() function 380		isspace 147
_VBDY macro 6		isupper 147
_wcsicmp() function 413	C	isxdigit 147
_wcsnicmp() function 415	calculating	wide alphabetic character 150
	absolute value 38	wide alphanumeric character 150
•	absolute value of long integer 155	wide control character 150
A	arccosine 39	wide decimal-digit character 150
abnormal program end 37	arctangent 45	wide hexadecimal digit 150
abort() function 37	base 10 logarithm 166	wide lowercase character 150
abs() function 38	cosine 64	wide non-alphanumeric
absolute value	error functions 72	character 150
abs() function 38	exponential function 73	wide non-space character 150
fabs 74	floating-point absolute value 74	wide printing character 150

character testing (continued)	converting (continued)	eofile
wide uppercase character 150	wide character to byte 435	clearing 245
wide whitespace character 150	wide character to long integer 424	macro 15
character testing functions 34	wide character to multibyte	resetting error indicator 62
clear error indicators 62	character 436	erf() function 72
clearerr 62	wide-characterc string to unsigned	erfc() function 72
clock() function 63	long 430	errno 4
	<u> </u>	
CLOCKS_PER_SEC 63	zoned decimal to double 224	errno.h include file 4
closing	zoned decimal to integer 225	errno macros 451
file 229	copying	errno values for Integrated File
message catalog 57	bytes 189, 191	System 453
stream 75	strings 330, 346	errno variable 198
comparing	cos() function 64	error handling
buffers 188	cosh() function 65	assert 44
strings 326, 329, 331, 344	creating	clearerr 62
comparing strings 323, 341	a temporary file 374, 375	ferror 79
compile regular expression 237	ctime() function 66	functions 21
concatenating strings 324, 343	ctime_r() function 67	perror 198
conversion functions	ctype functions 147	stream I/O 79
QXXDTOP 219	ctype.h include file 3	strerror 333
QXXDTOZ 220	currency functions 24	error indicator 79
QXXITOP() 221		error macros, mapping stream I/O
QXXITOZ 221		exceptions 455
QXXPTOD 222	D	error messages
	ט	ĕ
QXXPTOI 223	data conversion	printing 198
QXXZTOD 224	atof() function 47	except.h include file 4
QXXZTOI 225	atoi() function 49	exception class
converting	atol() function 50	listing 461
character case 377	data items 108	mapping 458
character string to double 357	data type compatibility	exit() function 73
character string to long integer 362	CL 465, 467, 468	EXIT_FAILURE 16, 73
date 350	COBOL 466	EXIT_SUCCESS 16, 73
double to zoned decimal 220	ILE COBOL 463	exp() function 73
floating-point numbers to integers and		exponential functions
fractions 195	RPG 462, 465	exp 73
floating point to packed decimal 219	data type limits 7	frexp 112
from structure to string 40	date and time conversion 350	ldexp 156
from structure to string (restartable	decimal.h include file 4	log 165
version) 42	deleting	log10 166
integer to a character in the ASCII	file 243	pow 199
6	record 232, 243	sqrt 318
charactger set 376	determine the display width of a wide	sqrt 316
integer to packed decimal 221	character 441	
integer to zoned decimal 221	determining	_
local time 193	display width of a wide character	F
monetary value to string 334	string 433	fabs() function 74
multibyte character to a wide	display width of a wide-character	fclose() function 75
character 176	string 434	fdopen() function 76
multibyte character to wchar_t 186	length of a multibyte character 173	feof() function 79
multibyte string to a wide character	· ·	
string 180	differential equations 23	ferror() function 79
packed decimal to double 222	difftime() function 69	fflush() function 80
packed decimal to integer 223	divf() function 70	fgetc() function 82
single byte to wide character 54		fgetpos() function 84
sting to formatted date and time 407	_	fgets() function 85
string segment to unsigned	E	fgetwc() function 86
integer 364	end-of-file indicator 62, 79	fgetws() function 88
strings to floating-point values 47	ending a program 37, 73	file
	environment	appending to 76
strings to integer values 49		handle 90
strings to long values 50	functions 32	include 3
time 141, 143, 163, 164, 350	interaction 32	maximum opened 15
time to character string 66, 67	retrieving information 158	name length 15
wide character case 379	table 135	positioning 245
wide-character string to double 422	variables 135, 211	renaming 244
wide character string to multibyte	environment variables	updating 76
string 418	adding 211	file errors 62
wide character to a multibyte	changing 211	
character 396	searching 135	file handling
	S .	remove 243

file handling (continued)	include files	ldiv() function 156
rename 244	assert.h 3	length function 341
tmpnam 375	ctype.h 3	length of variables 462
file name length 15	decimal.h 4	library functions
file names, temporary 15	errno.h 4	absolute value
file positioning 84, 114, 117, 118	except.h 4	abs 38
FILE type 15	float.h 7	fabs 74
fileno() function 90	limits.h 7	labs 155
*		
float.h include file 7	locale.h 7	character case mapping
floor() function 91	math.h 8	tolower 377
flushing buffers 80	pointer.h 9	toupper 377
fmod() function 91	recio.h 9	towlower 379
fopen() function 92	regex.h 13	towupper 379
fopen, maximum simultaneous files 15	setjmp.h 13	character testing
format data as wide characters 123	signal.h 14	isalnum 147
formatted I/O 99	stdarg.h 14	isalpha 147
fpos_t 16	stddef.h 14	isascii 146
1	stdio.h 15	iscntrl 147
fprintf() function 99		
fputc() function 101	stdlib.h 16	isdigit 147
fputs() function 103	string.h 17	isgraph 147
fputwc() function 104	time.h 17	islower 147
fputws() function 106	wcstr.h 18	isprint 147
fread() function 108	xxcvt.h 18	ispunct 147
free() function 109	xxdtaa.h 19	isspace 147
freopen() function 111	xxenv.h 19	isupper 147
frexp() function 112	xxfdbk.h 19	iswalnum 150
fscanf() function 113		iswalpha 150
	indicators, error 62	1
fseek() function 114	initial strings 346	iswcntrl 150
fseeko() function 114	integer	iswctype 152
fsetpos() function 117	pseudo-random 227	iswdigit 150
ftell() function 118	Integrated File System errno values 453	iswgraph 150
fwide() function 120	internationalization 7	iswlower 150
fwprintf() function 123	interrupt signal 313	iswprint 150
fwrite() function 127	invariant character	iswpunct 150
fwscanf() function 128	hexadecimal representation 469	iswspace 150
TWO CHILLY THE CONTROL TEC	isalnum() function 147	iswupper 150
		1.1
	isalpha()function 147	iswxdigit 150
G	isascii() function 146	isxdigit 147
gamma() function 131	isblank() function 149	conversion
getc() function 133	iscntrl() function 147	QXXDTOP 219
getchar() function 133	isdigit() function 147	QXXDTOZ 220
9	isgraph() function 147	QXXITOP 221
getenv() function 135	islower() function 147	QXXITOZ 221
gets() function 137	isprint() function 147	OXXPTOD 222
getting	ispunct() function 147	QXXPTOI 223
handle for character mapping 437	isspace()function 147	QXXZTOD 224
handle for character property	isupper() function 147	QXXZTOI 225
classification 438	11 "	
wide character from stdin 140	iswalnu() function 150	strfmon 334
getwc() function 138	iswcntrl() function 150	strptime 350
getwchar() function 140	iswctype() function 152	wcsftime 407
gmtime() function 141	iswdigit() function 150	data areas
	iswgraph() function 150	QXXCHGDA 218
gmtime_r() function 143	iswlower() function 150	QXXRTVDA 223
	iswprint() function 150	error handling
	iswpunct() function 150	_GetExcData 135
Н	iswspace() function 150	clearerr 62
handling interrupt signals 313	iswupper() function 150	raise 226
HUGE_VAL 8		
	iswxdigit() function 150	strerror 333
hypot() function 145	isxdigit() function 147	exponential
hypotenuse 145		exp 73
		frexp 112
_	L	ldexp 156
1	labe() function 155	log 165
- L/O amono 62	labs() function 155	log10 166
I/O errors 62	langinfo.h include file 7	pow 199
idate	language collation string	file handling
correcting for local time 163, 164	comparison 404	fileno 90
functions 24	ldexp() function 156	

library functions (continued) file handling (continued)	library functions (continued) multibyte	library functions (continued) stream input/output (continued)
remove 243	_wcsicmp 413	fscanf 113
rename 244	_weshirip 415	fseek 114
tmpfile 374	btowc 54	fsetpos 117
tmpnam 375	mblen 172	ftell 118
locale	mbrlen 173	fwide 120
localecony 158	mbrtowc 176	fwprintf 123
nl_langinfo 196	mbsinit 179	fwrite 127
setlocale 308	mbsrtowcs 180	fwscanf 128
strxfrm 366	mbstowcs 182	getc 133
math	mbtowc 186	getchar 133
acos 39	towctrans 378	gets 137
asin 43	wcrtomb 396	getwc 138
atan 45	wcscat 400	getwchar 140
atan2 45	weschr 402	printf 200
bessel 51	wescmp 403	putc 210
ceil 61	wcscoll 404	putchar 210
cos 64	wcscpy 405	puts 212
cosh 65	wcscspn 406	putwc 213
div 70	wcslen 409	putwchar 214
erf 72	wcsncat 410	scanf 299
erfc 72	wesnemp 411	setbuf 305
floor 91	-	setvbuf 311
fmod 91	wcsncpy 413 wcspbrk 416	sprintf 317
	-	=
frexp 112	wcsrchr 417	sscanf 322
gamma 131	wcsrtombs 418	swprintf 367
hypot 145	wcsspn 420	swscanf 369
ldiv 156	wcstombs 426	ungetc 381
log 165	wcswcs 432	ungetwc 382
log10 166	wcswidth 433	vfprintf 386
modf 195	wcsxfrm 434	vfwprintf 387
sin 315	wctob 435	vprintf 389
sinh 316	wctomb 436	vsnprintf 390
sqrt 318	wctrans 437	vsprintf 391
tan 371	wctype 438	vswprintf 393
tanh 372	wcwidth 441	vwprintf 394
memory management	program	wprintf 447
calloc 56	abort 37	wscanf 448
free 109	atexit 46	string manipulation
malloc 170	exit 73	strcat 324
realloc 234	signal 313	strchr 325
memory operations	ĕ	
, 1	regular expression	strcmp 326 strcoll 329
memchr 187	regcomp 237	
memcmp 188	regerror 238	strcpy 330
memcpy 189	regexec 240	strcspn 331
memmove 191	regfree 242	strlen 341
memset 192	searching	strncmp 344
wmemchr 442	bsearch 52	strncpy 346
wmemcmp 443	qsort 216	strpbrk 348
wmemcpy 444	stream input/output	strrchr 354
wmemmove 445	fclose 75	strspn 355
wmemset 446	feof 79	strstr 356
message catalog	ferror 79	strtod 357
catclose 57	fflush 80	strtok 359
catgets 58	fgetc 82	strtok_r 361
catopen 59	fgetpos 84	strtol 362
miscellaneous	fgets 85	strtoul 364
assert 44	fgetwc 86	strxfrm 366
getenv 135	fgetws 88	wcsstr 421
	O .	wcstok 429
longjmp 168	fprintf 99	
perror 198	fputc 101	time
putenv 211	fputs 103	asctime 40
rand 227	fputwc 104	asctime_r 42
rand_r 227	fputws 106	clock 63
setjmp 307	fread 108	ctime 66
srand 319	freopen 111	ctime_r 67
	-	

library functions (continued)	logic errors 44	memory operations
time (continued)	logical record length 94	memchr 187
difftime 69	longjmp() function 168	memcmp 188
gmtime 141	lrecl 94	memcpy 189
gmtime_r 143		memmove 191
localtime 163		memset 192
localtime_r 164	M	wmemchr 442
mktime 193	IVI	wmemcmp 443
strftime 336	malloc() function 170	wmemcpy 444
	math functions	
wcsftime 407	abs 38	wmemmove 445
trigonometric	acos 39	wmemset 446
acos 39	asin 43	memset() function 192
asin 43	atan 45	miscellaneous functions
atan 45	atan2 45	assert 44
atan2 45	bessel 51	getenv 135
cos 64		longjmp 168
cosh 65	div 70	perror 198
sin 315	erf 72	putenv 211
sinh 316	erfc 72	rand 227
tan 371	exp 73	rand_r 227
tanh 372	fabs 74	setjmp 307
type conversion	floor 91	srand 319
,	fmod 91	
atof 47	frexp 112	mktime() function 193
atoi 49	gamma 131	modf() function 195
atol 50	hypot 145	monetary functions 24
strol 362	labs 155	monetary.h include file 8
strtod 357	ldexp 156	multibyte functions
strtoul 364	ldiv 156	_wcsicmp 413
toascii 376		_wcsnicmp 415
wcstod 422	log 165	btowc 54
wcstol 424	log10 166	mblen 172
wcstoul 430	modf 195	mbrlen 173
variable argument handling	pow 199	mbrtowc 176
va_arg 384	sin 315	mbsinit 179
va_end 384	sinh 316	mbsrtowcs 180
va_start 384	sqrt 318	mbstowcs 182
vfprintf 386	tan 371	mbtowc 186
vprintf 389	tanh 372	towetrans 378
	math.h include file 8	
vsnprintf 390	mathematical functions 22	wcrtomb 396
vsprintf 391	maximum	wcscat 400
library introduction 21	file name 15	wcschr 402
limits.h include file 7	opened files 15	wcscmp 403
llabs() subroutine	temporary file name 15	wcscoll 404
absolute value of long long	MB CUR MAX 16	wcscpn 406
integer 155	mblen() function 172	wcscpy 405
lldiv() subroutine		wcscspn 406
perform long long division 156	mbrlen() function 173	wcsicmp 413
local time corrections 163	mbrtowc() function 176	wcslen 409
local time corrections (restartable	mbsinit() function 179	wcsncat 410
version) 164	mbsrtowcs() function 180	wcsncmp 411
locale functions	mbstowcs() function 182	wcsncpy 413
localecony 158	mbtowc() function 186	wcsnicmp 415
setlocale 308	memchr() function 187	weshlenip 116 wespbrk 416
strxfrm 366	memcmp() function 188	wesperk 116 wesrchr 417
locale.h include file 7	memcpy() function 189	western 417 westernbs 418
	memicmp() function 190	
localeconv() function 158	memmove() function 191	wcsspn 420
locales	memory allocation	wcstombs 426
retrieve information 196	calloc 56	wcswcs 432
setting 308	free 109	wcswidth 433
localtime() function 163	malloc 170	wcsxfrm 434
localtime_r() function 164	realloc 234	wctob 435
locating storage 109	memory management	wctomb 436
log() function 165	calloc 56	wctrans 437
log10() function 166	free 109	wctype 438
logarithmic functions		wcwidth 441
log 165	malloc 170 realloc 234	
log10 166		
	memory object functions 32	

NI.	random access 114, 118	regex.h include file 13
N	random number generator 227, 319	regexec() function 240
NDEBUG 3, 44	read operations	regfree() function 242
nl_langinfo() function 196	character from stdin 133	remove() function 243
nltypes.h include file 9	character from stream 133	rename() function 244
nonlocal goto 168, 307	data items from stream 108	reopening streams 111
NULL pointer 14, 15, 16	formatted 113, 299, 322	reserving storage
	line from stdin 137	malloc 170
0	line from stream 85	realloc 234
	reading a character 82	retrieve data area 223
offsetof macro 14	scanning 113	retrieve locale information 196
opening	reading character 133	rewind() function 245
message catalog 59	data 299	
	data from stream using wide	S
P	character 128	
F	data using wide-character format	scanf() function 299
passing	string 448	searching
constants 468	formatted data 113	bsearch function 52
variables 468	items 108	environment variables 135
perror() function 198	line 137	strings 325, 348, 355 strings for tokens 359, 361
pointer.h include file 9	messages 58	searching and sorting functions 22
pow() function 199	stream 85	seed 319
printf() function 200 printing	wide character from stream 86, 138	send signal 226
error messages 198	wide-character string from stream 88	separate floating-point value 112
process control	reallocation 234	setbuf() function 305
signal 313	reallocation 234 recfm 94	setjmp() function 307
program termination	recio.h include file 9	setjmp.h include file 13
abort 37	record format 94	setlocale() function 308
atexit 46	record input/ouput	setting
exit 73	_Racquire 228	bytes to value 192
pseudo-random integers 227	_Rclose 229	setvbuf() function 311
pseudorandom number functions	_Rcommit 230	signal() function 313
rand 227	_Rdelete 232	signal.h include file 14 signal handling 456
rand_r 227 srand 319	_Rdevatr 233	sin() function 315
ptrdiff_t 14	_Rfeod 246	sine 315
pushing characters 381	_Rfeov 247	sinh() function 316
putc() function 210	_Rformat 249	size_t 14
putchar() function 210	_Rindara 251 _Riofbk 252	snprintf() function 320
putenv() function 211	_Rlocate 254	sorting
puts() function 212	_Ropen 257	quick sort 216
putwc() function 213	_Ropnfbk 261	sprintf() function 317
putwchar() function 214	_Rpgmdev 262	sqrt() function 318
	_Rreadd 263	srand() function 319
	_Rreadf 265	sscanf() function 322
Q	_Rreadindv 267	standard types FILE 15
qsort() function 216	_Rreadk 270	stdarg.h include file 14
quick sort 216	_Rreadl 274	stddef.h include file 14
QXXCHGDA() function 218	_Rreadn 275	stdio.h include file 15
QXXDTOP() function 219	_Rreadnc 278	stdlib.h include file 16
QXXDTOZ() function 220	_Rreadp 279	stopping
QXXITOP() function 221	_Rreads 282 Rrelease 283	program 37
QXXITOZ() function 221	_Rrlslck 285	storage allocation 56
QXXPTOD() function 222	_Rrollbck 286	strcasecmp() function 323
QXXPTOI() function 223 QXXRTVDA() function 223	_Rupdate 288	strcat() function 324
QXXXTOD() function 224	_Rupfb 290	strchr() function 325
QXXZTOB() function 225	_Rwrite 291	strcmp() function 326
2.5.2.101() ranction 220	_Rwrited 293	stronpi() function 328
	_Rwriterd 296	strony() function 329
R	_Rwrread 297	strcpy() function 330 strcspn() function 331
	record program ending 46	strdup() function 332
raise() function 226 rand() function 227	redirection 111	stream I/O functions 27
RAND_MAX 16	regcomp() function 237	stream input/output
rand_r() function 227	regerror() function 238	fclose 75

stream input/output (continued)	streams (continued)	strtok_r() function 361
feof 79	ungetting characters 381	strtol() function 362
ferror 79	updating 92, 111	strtoll() subroutine
fflush 80	writing characters 101, 210	character string to long long
fgetc 82	writing data items 127	integer 362
fgets 85	writing lines 212	strtoul() function 364
fopen 92	writing strings 103	strtoull() subroutine
fprintf 99	strerror() function 333	character string to unsigned long long
	**	
fputc 101	strfmon() function 334	integer 364
fputs 103	strftime() function 336	strxfrm() function 366
fputwc 104	stricmp() function 340	swprintf() function 367
fputws 106	string.h include file 17	swscanf() function 369
fread 108	string manipulation	system() function 370
freopen 111	strcasecmp 323	
fscanf 113	strcat 324	_
fseek 114	strchr 325	T
ftell 118	strcmp 326	
fwrite 127	strcoll 329	tan() function 371
getc 133	strcpy 330	tangent 371
getchar 133	strcspn 331	tanh() function 372
gets 137	strlen 341	testing
0		ASCII value 146
printf 200	strncasecmp 341	character property 149, 152
putc 210	strncat 343	isalnum 147
putchar 210	strncmp 344	isalpha 147
puts 212	strncpy 346	iscntrl 147
rewind 245	strpbrk 348	isdigit 147
scanf 299	strrchr 354	isgraph 147
setbuf 305	strspn 355	
setvbuf 311	strstr 356	islower 147
snprintf 320	strtod 357	isprint 147
sprintf 317	strtok 359	ispunct 147
sscanf 322	strtok_r 361	isspace 147
swprintf 367	strtol 362	isupper 147
swscanf 369	strxfrm 366	isxdigit 147
		state object for initial state 179
tmpfile 374	wcsstr 421	wide alphabetic character 150
ungetc 381	wcstok 429	wide alphanumeric character 150
ungetwc 382	strings	wide control character 150
va_arg 384	comparing 331, 344	wide decimal-digit character 150
va_end 384	concatenating 324	wide hexadecimal digit 150
va_start 384	converting	wide lowercase character 150
vfprintf 386	to floating-point 50	
vfwprintf 387	to integer 49	wide non-alphanumeric
vprintf 389	to long integer 50	character 150
vsnprintf 390	copying 330	wide non-space character 150
vsprintf 391	ignoring case 326, 329, 331	wide printing character 150
vswprintf 393	initializing 346	wide uppercase character 150
vwprintf 394	length of 341	wide whitespace character 150
wprintf 447	reading 85	testing state object for initial state 179
wscanf 448	searching 325, 348, 355	time
	<u> </u>	asctime 40
stream orientation 120	searching for tokens 359, 361	asctime_r 42
streams	strstr 356	converting from structure to
access mode 111	writing 103	string 40
appending 92, 111	strlen() function 341	converting from structure to string
binary mode 111	strncasecmp() function 341	(restartable version) 42
buffering 305	strncat() function 343	correcting for local time 163, 164
changing current file position 114,	strncmp() function 344	,
118	strncpy() function 346	ctime 66
changing file position 245	strnicmp() function 347	ctime_r 67
formatted I/O 113, 200, 299, 317, 322	strnset() function 349	difftime 69
opening 92	strpbrk() function 348	function 163, 164
reading characters 82, 133	strptime() function 350	functions 24
reading data items 108	strrchr() function 354	gmtime 141
ĕ		gmtime_r 143
reading lines 85, 137	strset() function 349	localtime 163
reopening 111	strspn() function 355	localtime_r 164
rewinding 245	strstr() function 356	mktime 193
text mode 111	strtod() function 357	strftime 336
translation mode 111	strtok() function 359	time 373

time() function 373	W	X
time.h include file 17	1 1 1 1 61 40	
tm structure 141, 143	wchar.h include file 18	xxcvt.h include file 18
TMP_MAX 375	wcrtomb() function 396	xxdtaa.h include file 19
tmpfile() function	wcscat() function 400	xxenv.h include file 19
names 15	wcschr() function 402	xxfdbk.h include file 19
number of 15	wcscmp() function 403	
tmpnam() function	wcscoll() function 404	
file names 15	wcscpy() function 405	
tmpnam() 375	wcscspn() function 406	
toascii() function 376	wcsftime() function 407	
tokens	wcslen() function 409	
strtok 359	wcsncat() function 410	
strtok_r 361	wcsncmp() function 411	
tokenize string 359	wcsncpy() function 413	
tolower() function 377	wcspbrk() function 416	
toupper() function 377	wcsrchr() function 417	
towetrans() function 378	wcsrtombs() function 418	
towlower() function 379	wcsspn() function 420	
towupper() function 379	wcsstr() function 421	
trigonometric functions	wcstod() function 422	
acos 39	wcstok() function 429	
asin 43	wcstol() function 424	
atan 45	wcstoll() subroutine	
atan2 45	wide character to long long	
cos 64	integer 424	
cosh 65	wcstombs() function 426	
sin 315	wcstoul() function 430	
sinh 316	wcstoull() subroutine	
tan 371	wide-character string to unsigned long	
tanh 372	long 430	
type conversion	wcstr.h include file 18	
atof 47	wcswcs() function 432	
atoi 49	wcswidth() function 433	
atol 50	wcsxfrm() function 434	
strtod 357	wctob() function 435	
strtol 362	wctomb() function 436	
strtoul 364	wctrans() function 437	
toascii 376	wctype() function 438	
wcstod 422	wctype.h include file 18	
wcstol 424	wewidth() function 441	
wcstoul 430	wide character string functions 35	
	wmemchr() function 442	
	wmemcmp() function 443	
U	wmemcpy() function 444	
ungetc() function 381	wmemmove() function 445	
ungetwc() function 382	wmemset() function 446	
updating files 76	wprintf() function 447	
updating mes 70	write operations	
	character to stdout 101, 210	
V	character to stream 101, 210, 381	
•	data items from stream 127	
va_arg() function 384	formatted 99, 200, 317	
va_end() function 384	line to stream 212 printing 127	
va_start() function 384	1 0	
variable argument functions 31	string to stream 103	
verify condition 44	writing character 101, 210	
vfprintf() function 386		
vfwprintf() function 387	data items from stream 127	
vprintf() function 389	formatted data to a stream 99	
vsnprintf() function 390	string 103, 212	
vsprintf() function 391	wide character 104, 213, 214	
vswprintf() function 393	wide-character string 106 wide characters to a stream 123	
vwprintf() function 394	wide characters to a stream 123 wscanf() function 448	
	wacamy function 440	

Readers' Comments — We'd Like to Hear from You

iSeries ILE C/C++ for iSeries Run-Time Library Functions Version 5

Phone No.

V 0151011 0					
Publication No. SC41-560	07-01				
Overall, how satisfied are	e you with the info	ormation in this	book?		
	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction					
How satisfied are you that	at the information	in this book is:			
	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate Complete Easy to find Easy to understand Well organized Applicable to your tasks					
Please tell us how we can	improve this boo	ok:			
Thank you for your respo	nses. May we conta	act you? 🗌 Ye	s 🗌 No		
When you send comments way it believes appropriat				or distribute your o	comments in any
Name		Ac	ldress		
Company or Organization	1				

Readers' Comments — We'd Like to Hear from You SC41-5607-01



Cut or Fold Along Line

Fold and Tape

Please do not staple

Fold and Tape



NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM CORPORATION ATTN DEPT 542 IDCLERK 3605 HWY 52 N ROCHESTER MN 55901-7829



Fold and Tape

Please do not staple

Fold and Tape

IBM.



Printed in the United States of America on recycled paper containing 10% recovered post-consumer fiber.

SC41-5607-01

